

THESE

Présentée devant

L'UNIVERSITE DE RENNES 1

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITE DE RENNES 1

Mention : INFORMATIQUE

PAR

Marouane HIMDI

Equipe d'accueil : TRISKELL (IRISA)

Ecole Doctorale : MATISSE

Composante universitaire : IFSIC

PERFORMANCES DES SYSTEMES DISTRIBUES : PROPOSITION D'UNE PLATEFORME FEDERATRICE DE LA SUPERVISION

SOUTENUE LE **20 décembre 2007** devant la commission d'Examen

COMPOSITION DU JURY :

M. César VIHO	Président
Mme. Chantal ROBACH	Rapporteur
Mme. Isabelle BORNE	Rapporteur
M. Jean-Marc JEZEQUEL	Directeur de thèse
M. Yves LE TRAON	Encadrant
M. Alain RIBAUT	Examineur (KEREVAL)

Dédicaces

Je dédie ce travail

A

Mes chers parents, Larbi et Jamila, qui ont toujours été là pour moi et qui m'ont donnés et me donnent un magnifique modèle de labour et de persévérance

A

Kaouthar, mon épouse, pour avoir accepté tant de sacrifices durant les années de la thèse

A

Ma fille Rim et mes futurs enfants, qu'ils sachent que la réussite est dans le travail

A

Mes frères et sœurs, Bouchra, Hassan, Bahija, Mohammed-Ali et Assmaa ainsi que leurs adorables enfants. Merci pour vos encouragements au quotidien

A

Ma belle famille en témoignage de remerciements

A

Toute ma famille

Remerciements

Me voilà livré à cet exercice difficile de rédiger ces lignes de remerciements. Difficile ! Et bien oui, car ils sont nombreux les personnes qui ont contribué, de près ou de loin, à la réalisation de ce travail de thèse. Allons donc à l'essentiel, et efforçons-nous de n'oublier personne.

Je tiens tout d'abord à remercier M. César VIHO de l'équipe Armor à l'IRISA, qui m'a fait l'honneur de présider mon jury de thèse. Toute ma gratitude s'adresse également à mes deux rapporteurs : Mme Isabelle BORNE de l'université de Bretagne Sud dont la lecture extrêmement approfondie de ce manuscrit a permis d'en améliorer la qualité et à Mme Chantal ROBACH de l'INPG.

Un grand merci à mon directeur de thèse Jean-Marc JEZEQUEL et mon encadrant Yves LeTRAON pour les multiples conseils et le suivi lors de la rédaction de ce manuscrit. Bien que leurs tâches n'a pas été faciles vu le contexte industriel de cette thèse, ils ont su intervenir aux moments opportuns pour que ce travail aille jusqu'au bout. Je saisis l'occasion pour remercier toute l'équipe TRISKELL pour les conseils et les multiples relectures : Franck, Reda, Erwan, Loïc, Jean-Marie, Noël, Benoît, Vincent, François, Cyril, Olivier, Romain, Sébastien et tous ceux que j'ai oublié.

Cette thèse a été financée dans le cadre d'une convention CIFRE avec l'entreprise KEREVAL, une jeune entreprise innovante qui opère dans le domaine d'ingénierie et de test des logiciels. Je remercie chaleureusement Abdelmoula TAMOUDI, PDG de cette société pour tout le soutien qu'il m'a accordé. Un grand merci particulier à Alain RIBAUT mon encadrant à l'entreprise pour les différents échanges, pour son sens de l'humour et sa disponibilité malgré un agenda chargé. Je remercie également Annick, l'assistance de direction, toujours disponible pour répondre à mes questions administratives.

J'adresse mes sincères remerciements et ma reconnaissance à tout ceux et celles que j'ai côtoyé et que je continue à côtoyer à KEREVAL : Thierry (dit le cousin), François (dit Kyzdra), Jean-Marie (le foot-balleur), Sébastien, Mokrane, Vincent^{cube}, Grégory, Christophe, Alban, Julian, Bertrand, Raphaëlle, Mathieu, Franck et Kilian. Merci à vous tous pour les encouragements tout au long de la rédaction de mon rapport de thèse.

Une reconnaissance particulière pour mon ami (plutôt un grand frère) Ahmed pour les différents services qu'il m'a rendu et pour les conseils judicieux. Je n'oublierai jamais mes autres amis que j'ai côtoyé à la cité : Mansour, Walid, Mohammed-ali, Lahbib, Ibrahim, Adel, Adil, Hassan, Ali, Ahmed, Yahya, Aïssa, Mahmoud, Rabie, Najib et tous ceux que j'ai oublié de citer.

A Said, Achraf (je te souhaite la même aventure), Aziz, Youness, Nourreddine et tous mes amis au Maroc.

Table des index

PARTIE 1 INTRODUCTION.....	11
A. Introduction.....	13
1. La performance des applications : un enjeu de taille	13
2. Les outils d'évaluation des performances.....	14
3. Plan du document	15
PARTIE 2 ETAT DE L'ART.....	17
B. Concepts de base de la qualité de service et de l'évaluation des performances.....	19
1. Qualité de service	19
1.1. Définitions	19
1.2. Gestion de la QoS	20
1.2.1. Composantes de la gestion de QoS	21
1.2.2. Niveaux d'intervention pour la gestion la QoS	22
1.2.3. Niveau réseau	22
1.2.4. Niveau système	22
1.2.5. Niveau intergiciel « middleware »	23
1.2.6. Niveau applicatif.....	24
1.3. Synthèse.....	24
2. Concepts de l'évaluation des performances systèmes	26
2.1. Définitions	26
2.2. Intérêts de l'évaluation des performances système.....	26
2.3. Les techniques d'évaluation des performances.....	27
3. Besoins de l'industrie	28
3.1. Performance de la plate-forme de supervision.....	28
3.1.1. Intrusivité des sondes	28
3.1.2. Masse d'information	32
3.1.3. Synthèse	34
3.2. L'intégration des sondes hétérogènes	34
3.2.1. Introduction.....	34
3.2.2. Contrainte d'hétérogénéité	35
3.2.3. Synthèse	40
3.3. Découplage entre la collecte et la présentation	40
3.3.1. Introduction.....	40
3.3.2. Intérêt du découplage	41
4. Conclusion.....	45
C. Outils et techniques de mesure des performances systèmes.....	47
1. Standard SNMP	48
1.1. Principe.....	48
1.2. MIB.....	48
1.3. Atouts et faiblesses du standard SNMP	49
1.4. Discussion.....	50
2. Eclipse TPTP	50
2.1. Introduction.....	50
2.2. Architecture de TPTP	51
2.3. Atouts et faiblesses de la plate-forme TPTP	52
2.4. Discussion.....	53
3. NAGIOS	53
3.1. Présentation.....	53
3.2. Architecture	54
3.3. Atouts et faiblesses de la plate-formeNAGIOS	55
3.4. Discussion.....	56
4. LeWYS (Lewys is Watching Your System).....	56
4.1. Présentation.....	56
4.2. Architecture	57
4.3. Atouts et faiblesses de la bibliothèque LeWYS.....	59

4.4.	Discussion.....	59
5.	Synthèse.....	60
PARTIE 3 CONCEPTION & MISE EN OEUVRE		63
D.	Conception.....	65
1.	Rappel de la problématique	65
2.	Description du framework de supervision	65
2.1.	Introduction.....	65
2.2.	Architecture physique	66
2.3.	Déroulement d'une supervision	67
2.4.	Architecture logicielle de la plate-forme	72
2.5.	Description des composants de la plate-forme.....	73
2.5.1.	Les sondes	73
2.5.2.	Moteur de traitement.....	74
3.	Conclusion.....	76
E.	Mise en oeuvre.....	79
1.	Introduction	79
2.	Implémentation d'une sonde	79
2.1.	Sonde Vmstat.....	79
2.1.1.	Description de la sonde	79
2.1.2.	Implémentation	81
2.2.	Autres exemples de sondes	82
3.	Implémentation du moteur de traitement.....	84
3.1.	L'analyseur « parseur ».....	84
3.2.	Structuration des informations.....	85
3.3.	Le module d'export « outputter »	86
3.4.	Création de l'adaptateur	87
4.	Fonctionnement de la plate-forme	89
4.1.	Tâche de transfert des fichiers	89
4.2.	Tâche de pilotage des sondes.....	90
4.3.	Tâche de traitement des fichiers de résultat	91
4.4.	Tâche de reporting	92
5.	Conclusion.....	94
PARTIE 4 EXPERIMENTATIONS.....		95
F.	Validation expérimentale	97
1.	Description du contexte de l'expérimentation	97
1.1.	Plate-forme expérimentale.....	97
1.2.	Démarche expérimentale	98
2.	Bilan de l'expérimentation	100
2.1.	Intrusivité au niveau des ressources systèmes	100
2.1.1.	Résultats.....	100
2.1.2.	Analyses.....	101
2.2.	Passage à l'échelle « scalability ».....	102
2.2.1.	Conclusions.....	105
2.3.	Intrusivité au niveau des ressources réseaux.....	105
2.3.1.	Cas TPTP	106
2.3.2.	Cas SNMP.....	108
2.3.3.	Conclusion	110
2.3.4.	Effet de l'intrusivité sur la bande passante.....	110
3.	Conclusion générale	113
G.	Cas d'utilisation de la plate-forme de supervision.....	115
1.	Premier exemple : Test de montée en charge	115
1.1.	Description de la plate-forme de test	115
1.2.	Déroulement du test	116
1.3.	Résultats et analyses	116
2.	Deuxième exemple : plate-forme d'observation réactive	118
2.1.	Description.....	118

2.2.	Mise en œuvre.....	118
PARTIE 5	CONCLUSIONS & PERSPECTIVES.....	119
H.	Conclusion et perspectives	121
1.	Conclusion.....	121
2.	Perspectives	122
PARTIE 6	ANNEXE	123
ANNEXE A.	Network Time Protocol.....	125
ANNEXE B.	Oids des ressources systèmes	128
ANNEXE C.	TPTP (Test & Performance Tools Platform).....	129
ANNEXE D.	NAGIOS.....	130
ANNEXE E.	FRACTAL & DREAM	131
ANNEXE F.	Format CBE (Common Based Event).....	134
ANNEXE G.	Les métriques collectées par Vmstat	135
ANNEXE H.	Performance d'une analyse statique Vs une analyse basée sur des règles	136
ANNEXE I.	Simple Event Correlator.....	137
PARTIE 7	BIBLIOGRAPHIE.....	139

Table des figures

Figure 1 : Niveaux structurels des systèmes informatiques	22
Figure 2 : Gestion de la mémoire sous Unix	26
Figure 3 : Circuit de traitement des données de supervision	33
Figure 4: Architecture classique d'une application web	35
Figure 5 : Extrait des fichiers générés respectivement par OpenSTA et Vmstat	37
Figure 6: Export des fichiers hétérogènes	38
Figure 7: Utilisation d'un format pivot pour la transformation	39
Figure 8 : Exemples de la syntaxe ULM et Xml-log	39
Figure 9: Principaux activités de la supervision	40
Figure 10 : Synchronisation par un médium	42
Figure 11 : Précision d'une synchronisation par un médium	43
Figure 12 : Synchronisation en aval	44
Figure 13 : Architecture du SNMP	48
Figure 14 : Objets de la MIB	49
Figure 15 : Architecture de TPTP	51
Figure 16 : Architecture de NAGIOS	54
Figure 17 : Des exemples d'une sonde composite et d'une sonde générique	57
Figure 18 : Exemple illustrant la reconfiguration dynamique	57
Figure 19 : Architecture d'une pompe	58
Figure 20 : Architecture JMX	59
Figure 21 : Architecture physique de la plate-forme de supervision	66
Figure 22 : Diagramme d'activités de la plate-forme de supervision	67
Figure 23 : Déroulement d'une supervision	68
Figure 24 : Mise en forme des fichiers de résultats	70
Figure 25 : Synchronisation par décalage temporel	70
Figure 26 : processus de génération des graphiques	71
Figure 27 : Exemple de corrélation entre plusieurs paramètres	72
Figure 28 : Diagramme de déploiement (avant supervision)	72
Figure 29 : Diagramme de déploiement (après supervision)	73
Figure 30 : Structure d'une sonde de la plate-forme	74
Figure 31 : Structure interne du « GLA »	75
Figure 32 : Syntaxe de la commande Vmstat	80
Figure 33 : Exemple des informations collectées avec Vmstat	80
Figure 34 : Implémentation de la sonde Vmstat	81
Figure 35 : Résultat généré par la sonde Vmstat	82
Figure 36 : Exemple de sortie de la commande Ifstat	83
Figure 37 : Structure interne du "GLA"	84
Figure 38 : Editeur Eclipse pour la création d'un « parseur »	85
Figure 39 : Structuration des résultats dans le format CBE	86
Figure 40 : Fichier de résultats Vmstat au format CSV	87
Figure 41 : Instanciation d'un adaptateur	87
Figure 42: Adaptateur générique pour la sonde Vmstat	88
Figure 43 : Tâche Ant pour le transfert local des fichiers	89
Figure 44 : Syntaxe d'utilisation de la commande pscp	89
Figure 45 : Tâche Ant pour le transfert distant des fichiers	90
Figure 46 : Syntaxe d'utilisation de la commande <i>plink</i>	90
Figure 47 : Tâche Ant pour le pilotage de la sonde Vmstat	91
Figure 48 : Déclaration des paramètres la sonde Vmstat	91
Figure 49 : Tâche Ant pour l'analyse des fichiers	92
Figure 50 : Exemple de graphique généré avec la bibliothèque JfreeChart	92
Figure 51 : Tâche Ant pour la génération et la corrélation des graphiques	92
Figure 52 : Plate-forme d'expérimentation	97
Figure 53 : Script PHP utilisé pour l'intrusivité	99
Figure 54 : Représentation « radar » des temps d'exécutions	101
Figure 55 : Passage à l'échelle - cas du SNMP	103
Figure 56 : Passage à l'échelle - cas de notre approche	104

Figure 57 : Passage à l'échelle - cas de TPTP	104
Figure 58 : Description des données applicatives de TPTP	107
Figure 59 : Formule approximative pour le calcul d'une réponse TPTP	107
Figure 60 : Volume de trafic généré par TPTP	108
Figure 61 : Calcul approximatif de la taille d'un échange SNMP	109
Figure 62 : Courbes des volumes de trafic générés par SNMP	109
Figure 63 : Volume du trafic généré par TPTP et SNMP	110
Figure 64 : Plate-forme de mesure de la bande passante.....	111
Figure 65 : Exécution de l'outil Iperf en mode serveur	111
Figure 66 : Effet de la supervision sur la bande passante.....	111
Figure 70 : Architecture NTP.....	125
Figure 71 : Trame NTP	126
Figure 72 : Diagramme de communication entre le client et le RAC	129
Figure 73 : Modèle de composant FRACTAL.....	131
Figure 74 : Le composant File de message DREAM.....	132
Figure 75 : Exemple d'un fichier de configuration SEC	137

Liste des tableaux

Tableau 1 : Synthèse des facteurs influant sur la QoS	25
Tableau 2 : Comparatif des solutions de supervision vis à vis des exigences	62
Tableau 3 : Liste des versions des modules installés sur la machine SUT.....	98
Tableau 4 : Mesures des temps d'exécution	100
Tableau 5 : Calcul des « overhead » et des gains relatifs	102
Tableau 6 : Détail des paquets échangés dans la plate-forme TPTP	106
Tableau 7 : Détail d'un échange SNMP	108
Tableau 8 : Echantillon de mesures effectuées avec Iperf	112
Tableau 9 : Comparaison des performances des "parseurs" dans le module GLA.....	136

Liste des équations

Équation 1 : Calcul de l'intrusivité au niveau du réseau.....	29
Équation 2 : Calcul de l'overhead d'une activité de supervision	31
Équation 3 : Calcul de l'overhead.....	101
Équation 4 : Calcul du gain relatif par rapport à SNMP	102
Équation 5 : Calcul du gain relatif par rapport à TPTP.....	102
Équation 6 : Approximation du volume de trafic généré par SNMP	109

Liste des acronymes

BER	Basic Encoding Rule
CBE	Component Based Event
CPU	Central Processing Unit
CSV	Comma-Separated Values
GLA	Generic Log Adapter
IETF	Internet Engineering Task Force
MIB	Management Information Bases
NRPE	Nagios Remote Plug-in Executor
NTP	Network Time Protocol
OASIS	Organization for the Advancement of Structured Information Standards
PDH	Performance Data Helper
QoS	Quality of Service
RAC	Remote Agent Controller
RMI	Remote Method Invocation
RPC	Remote Procedure Call
TPTP	Test & Performance Tools Platform
ULM	Uniform logger Messages
XML	eXtensible Markup Language

PARTIE 1
INTRODUCTION

A. Introduction

1. La performance des applications : un enjeu de taille

Le rôle essentiel de toute application informatique est d'accomplir les fonctions pour lesquelles elle a été conçue, et d'offrir une performance adaptée à un coût raisonnable. La performance d'une application est donc un facteur clé de son succès : un site Internet lent fera fuir ses visiteurs, un hébergeur proposant des services non performants verra ses clients partir vers la concurrence. Pire, pour une application stratégique, la lenteur peut réduire considérablement la productivité de tout un service ou entreprise, et engendrer des pertes substantielles.

La performance d'une application dépend de plusieurs facteurs, dont nombreux sont relatifs à une bonne conception de celle-ci. L'approche du génie logiciel préconise la séparation complète des parties logiques et physiques : une application conçue suivant les règles de l'art est une application dont les fonctionnalités ont été correctement abstraites, qui ne dépendent pas des spécificités des systèmes et dont les détails d'implémentation sont masqués. Cette démarche est en général associée à une approche descendante « Top-down »: de l'analyse des besoins vers l'implémentation physique.

Bien qu'avantageuse, l'approche descendante pose certains problèmes de compréhension. En effet, elle peut sous-entendre aux yeux des concepteurs que les ressources systèmes sont illimitées. La réalité est bien sûr différente; non seulement les ressources systèmes sont limitées par la puissance des machines mais, en plus, par le coût maximum fixé pour le développement et l'exploitation. Il en résulte une détection tardive des problèmes de performance et un traitement à chaud de ceux-ci, ce qui s'apparente dans la pratique à des interventions d'urgence visant à sauver les applications de la noyade via diverses « rustines », et qui finissent par rendre l'application difficilement maintenable.

La solution à ce problème revient à une prise en compte des enjeux et des aspects de la performance dès les premières phases de la conception des applications en utilisant différentes techniques d'évaluation des performances. Cette réflexion a donné naissance au concept des contrats de qualité du service (QoS) qui permettent de caractériser un service en tenant compte de son contexte d'exécution [Amorim.'04].

En plus de la définition des contrats de QoS d'un système, l'analyse des performances permet de disposer d'un profil de performance qui peut servir à identifier des problèmes de dimensionnement. Sans cela, le risque est de ne pas ou de mal identifier les goulots

d'étranglement réels du système. La tentation est alors de se fonder sur l'intuition, qui se révèle parfois trompeuse. En l'absence d'un profil de performance, des sommes d'argent peuvent être dépensées inutilement pour améliorer des parties du système qui n'ont que peu d'influence sur les performances globales de l'application. L'argent investi aura, alors, été gaspillé. C'est ce qu'IBM appelle l'e-panic (ou *epanic*) quand un problème de performance conduit quasi systématiquement les développeurs à ajouter des processeurs ou des lignes réseau sans que la cause du problème soit bien identifiée.

Que ce soit pour la définition des contrats de QoS ou la modélisation des performances d'un système, il est indispensable de disposer d'une plate-forme de supervision qui aide à la mesure des performances d'un système.

2. Les outils d'évaluation des performances

Vu la complexité des systèmes qu'on peut être amené à étudier (par exemple des systèmes répartis hétérogènes), la plate-forme de supervision doit s'adapter aux différentes configurations de ces systèmes. En général, les développeurs sont contraints de développer des outils de supervision *ad-hoc* pour collecter les informations nécessaires à l'évaluation des performances.

Le problème considéré dans cette thèse est celui de la construction d'une plate-forme fédératrice des outils de supervision. Cette plate-forme doit répondre aux exigences suivantes :

- ❖ *Faible intrusivité*

Il est impératif que la supervision ne perturbe que très peu les performances du système observé. Parmi les facteurs qui affectent les performances d'un outil de supervision on trouve le degré d'intrusivité des sondes de collecte.

- ❖ *Possibilité de découplage entre la phase de collecte et de présentation des informations*

Lorsqu'il s'agit d'évaluer plusieurs paramètres qui influent sur les performances d'un système (charge CPU, mémoires, trafic réseaux, etc.), il apparaît que les flux à considérer sont à la fois nombreux et complexes. Pour cela, il est très utile de pouvoir ajouter des traitements complexes comme le filtrage et la corrélation des informations afin de réduire la masse de données et ne garder que celles pertinentes.

- ❖ *La possibilité d'intégration des sondes hétérogènes*

Un exemple illustratif de cette exigence est le test en charge d'une application web où l'objectif est de vérifier si le système, sous les conditions décrites, est capable de fonctionner correctement et de connaître ses limites. Pour réaliser ce genre de test, il faut intégrer et corréler à la fois les informations fournies par l'outil de supervision (mesures de performances systèmes)

et celles fournies par l’outil de test en charge (temps de réponse, nombre d’utilisateurs, etc.). La difficulté dans cet exemple est liée à l’hétérogénéité et la distribution des systèmes observés. L’*hétérogénéité* implique que le framework de supervision doit être capable de gérer des informations de types différents et doit faciliter le développement de nouvelles sondes où l’intégration de sondes existantes. La *distribution* implique que les informations sont collectées sur des sites différents, éventuellement désynchronisés, ce qui rend difficile la corrélation de ces informations.

L’architecture de la plate-forme, que nous étudions, est composée d’un ensemble de sondes sélectionnées pour leurs faibles degrés d’intrusivité. Celles-ci enregistrent les informations, qu’elles collectent, dans des fichiers locaux. Ces derniers, sont traités par un composant générique d’analyse ayant pour rôle principal de mettre dans un format XML pivot le contenu de tous les fichiers et de faciliter leurs exploitations.

Nous avons développé un prototype de sonde pour mesurer les performances systèmes dans un environnement Unix. Nous avons ensuite comparé les performances de cette sonde, en terme de faible intrusivité, avec d’autres techniques de supervision telles que SNMP et la plate-forme Eclipse TPTP.

3. Plan du document

Ce manuscrit est organisé en cinq parties :

La première partie introduit la problématique traitée dans cette thèse. Il s’agit, rappelons le, d’une thèse à caractère industriel (contrat CIFRE) en partenariat avec la société KEREVAL qui est un laboratoire d’ingénierie et de test des logiciels. Du point de vue industriel, le sujet consiste en l’étude et la mise en œuvre d’une plate-forme pour fédérer des outils de supervision afin de mesurer les performances d’un système distribué. D’un point de vue scientifique, la difficulté est de trouver d’une part le bon degré d’abstraction et de modélisation pour permettre à des sondes de diverses natures d’inter-opérer, et d’autre part de faire les bonnes approximations (par exemple, dérive d’horloge négligée) pour obtenir une architecture de déploiement des sondes qui soit faiblement intrusive (engorgement réseau et ressources des machines).

La deuxième partie présente un état de l’art; elle est divisée en 2 chapitres :

- ❖ le premier chapitre présente les concepts de base de la qualité de service et des mesures de performance.

- ❖ le deuxième chapitre dresse un état de l'art des outils existants pour évaluer les performances d'un système et démontre les limites de ces outils et leurs inadéquations pour réaliser une supervision flexible et peu-intrusive.

La troisième partie décrit l'architecture de la plate-forme de supervision et se décompose en deux chapitres :

- ❖ le premier chapitre détaille l'aspect conception de cette plate-forme. Cette dernière est structurée en trois éléments principaux : les sondes qui assurent la fonction de collecte d'information, le moteur de traitement qui analyse les données collectées et enfin le composant de présentation qui affiche les informations sous un format exploitable par l'utilisateur.
- ❖ le deuxième chapitre décrit la mise en œuvre de la plate-forme sous forme d'un ensemble de tâches Ant. Cette approche de développement facilite l'intégration de la plate-forme de supervision dans d'autres frameworks qui font appel à une activité de supervision dans leurs fonctionnements comme c'est le cas des gestionnaires de test.

La quatrième partie est consacrée à valider expérimentalement la solution de supervision et à comparer ses performances par rapport à d'autres plates-formes ou standards de supervision et notamment la plate-forme TPTP d'Eclipse et SNMP. Nous démontrerons à travers cette comparaison que notre approche de supervision est moins intrusive, plus flexible et qu'elle facilite l'intégration et la prise en charge des résultats générés par des outils externes.

Enfin, la cinquième partie conclut ce rapport et présente des perspectives possibles de poursuite des travaux présentés dans ce manuscrit.

Mots clés

Systemes distribués, sondes de supervision, qualité de service, performances, plateforme générique, intrusivité, corrélation offline.

PARTIE 2
ETAT DE L'ART

B. Concepts de base de la qualité de service et de l'évaluation des performances

Les performances d'un système sont d'une importance capitale et constituent bien souvent la motivation première lors de l'élaboration d'une nouvelle solution ou d'une nouvelle implémentation. L'évaluation de la performance est l'une des composantes principales de l'évaluation de la Qualité de Service¹ (QoS) d'un système.

Le chapitre suivant définit les notions de bases en relation avec la qualité de service et l'évaluation des performances d'un système.

1. Qualité de service

1.1. Définitions

La qualité de service ne dispose pas d'une définition consensuelle qui conviendrait à tous les domaines auxquels elle s'applique. Cependant quel que soit le contexte, elle recouvre des propriétés extra-fonctionnelles (ex performances, sécurité, etc.) d'une entité informatique, d'un réseau, d'un middleware ou d'une application. La norme ISO² la définit comme *un ensemble de caractéristiques se rapportant au comportement collectif d'un ou plusieurs objets* [ISO.95].

C'est initialement pour les réseaux et télécommunications que la notion de QoS fut utilisée. Elle est alors généralement décrite par des critères quantitatifs tels que le délai, la gigue³. Il s'agit bien ici de propriétés extra-fonctionnelles qui décrivent les performances du réseau ou bien la qualité du service que le réseau rend aux applications qui l'utilisent.

Dans la même logique, il est donc opportun de définir la qualité de service d'une application vis-à-vis de l'utilisateur final. Dans cette perspective, il existe deux points de vue principaux qui mènent à deux définitions voisines mais cependant distinctes pour la qualité de service.

Le premier point de vue est celui du système ou plus exactement de l'environnement de déploiement de l'application et, la qualité de service est définie de la manière suivante pour les applications multimédias :

¹ QoS, Quality of Service, QoS en anglais

² International Standard Organization

³ Variation de délai

« *Quality of service represents the set of those quantitative and qualitative characteristics of a distributed MM4 system to achieve the required functionality of an application* »[Vogel, et al.'95]. Que nous traduirons par:

« *La qualité de service représente l'ensemble des caractéristiques quantitatives et qualitatives d'un système multimédia distribué qui permet d'atteindre la fonctionnalité requise pour une application.* ».

Cette définition introduit les critères qualitatifs et donc subjectifs liés à la caractérisation de la QoS. Cet aspect qualitatif ne peut être négligé car il décrit l'attente d'un utilisateur et permet d'évaluer la personnalisation à apporter au service pour le rendre acceptable.

Le second point de vue est justement celui de l'utilisateur :

« *Quality of service is the collective effect of service performance which determines the degree of satisfaction of a user of the service* »[CCITT.'89]

Que nous traduirons par « La qualité de service est l'effort collectif des performances du service qui détermine le degré de satisfaction de l'utilisateur du service. ». Cette satisfaction nécessite de connaître l'opinion de l'utilisateur et donc que celle-ci soit récoltée avant l'évaluation de cette même qualité de service.

Selon le point de vue adopté, la qualité de service regroupe donc les caractéristiques d'un système ou le résultat de ces caractéristiques sur les performances du système vis-à-vis du service rendu à l'utilisateur. Ces caractéristiques sont quantitatives ou qualitatives en fonction de l'angle sous lequel on les perçoit : une caractéristique comme le temps de réponse peut être mesurée de façon objective, en revanche, sa valeur en terme de qualité de service ne peut être définie qu'à l'aide des attentes de l'utilisateur.

La qualité de service peut être décrite par différents paramètres qu'il ne suffit pas de définir et de mesurer mais également de structurer afin d'appréhender les articulations. C'est ce que nous présentons dans le paragraphe suivant en proposant une hiérarchisation de certains paramètres influant sur la QoS.

1.2. Gestion de la QoS

La gestion de la QoS décrit l'ensemble des activités mises en place dans un système pour la surveiller et la contrôler [ISO.'97]. Ces activités nécessitent d'abord la définition d'un ensemble de composantes et des niveaux d'interventions pour gérer la QoS.

⁴ MultiMedia

1.2.1. Composantes de la gestion de QoS

Les trois composantes à définir dans la gestion de la QoS sont :

- *Le critère*: telle que nous l'avons défini au début de ce chapitre, la QoS recouvre l'ensemble des propriétés extra-fonctionnelles d'un système informatique. Chaque propriété définit un critère avec lequel la QoS peut être évalué. Ainsi, nous nous intéressons, dans cette thèse, au critère de performance auquel nous essayons de proposer un framework générique pour faciliter l'évaluation des performances d'un système informatique.
- *Les facteurs*: après avoir défini le critère, l'étape suivante dans la gestion de la QoS consiste à identifier les facteurs qui influent sur ce critère. Cet exercice nécessite une bonne maîtrise de l'architecture du service: Par exemple, les performances d'une application java dépendent de plusieurs paramètres dont le paramétrage de la machine virtuelle (par exemple les options Xmx et Xms⁵).
- *Les mesures*: l'étape de mesure permet de disposer d'un ensemble de données quantitatifs ou métriques sur la QoS. Ces données peuvent être utilisées pour comparer plusieurs configurations d'un système informatique et choisir celle qui offre la meilleure QoS. Le temps de réponse et la consommation des ressources systèmes sont des exemples parmi d'autres mesures qui permettent d'évaluer les performances d'un service. Cette activité de mesure nécessite des outils de supervision capables de collecter les informations, les analyser puis les présenter sous un format exploitable par l'utilisateur. L'étude de l'état de l'art des principales plates-formes de supervision existantes (Cf. Chapitre C : Outils et techniques de mesure des performances systèmes) ne nous a pas donné satisfaction car elles ne répondent pas complètement aux attentes de l'industrie qui cherche des outils à faible degrés d'intrusivité et qui offrent plus de flexibilité dans le traitement des informations collectées (Cf. paragraphe. 3 : Besoins de l'industrie, Page. 28). C'est dans ce contexte, que nos travaux se proposent d'apporter une solution de supervision pour combler ce manque d'un framework générique à faible intrusivité qui offre la possibilité de collecter, d'analyser et de corréler les mesures liées aux performances des systèmes.

⁵ Xms et Xmx sont des paramètres java qui permettent de dimensionner l'espace mémoire utilisée par l'application.

1.2.2. Niveaux d'intervention pour la gestion la QoS

La gestion de la QoS peut être mise en œuvre aux différents niveaux structurels des systèmes informatiques: réseau, système, middleware et application (Cf. Figure 1)

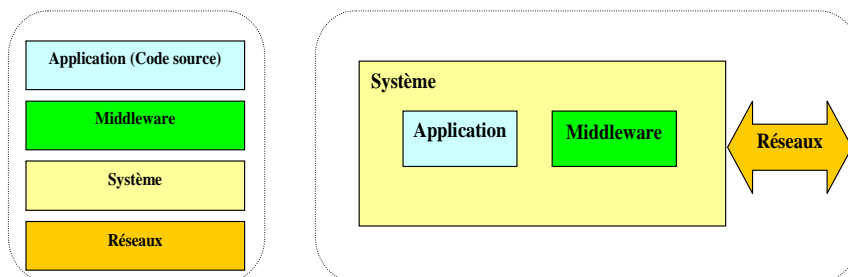


Figure 1 : Niveaux structurels des systèmes informatiques

1.2.3. Niveau réseau

La QoS au niveau du réseau peut être caractérisée par la mesure d'un ensemble de paramètres qui permettent d'évaluer la performance globale d'un réseau. Parmi ces paramètres on trouve :

- **La bande passante** (en anglais bandwidth): elle caractérise la vitesse avec laquelle les données peuvent transiter sur le réseau.
- **La latence**, délai (en anglais Delay) : elle caractérise le retard entre l'émission et la réception d'un paquet.
- **Le gigue** (en anglais jitter) : elle représente la variation du délai.
- **Perte de paquet** (en anglais packet loss): elle correspond à la non-délivrance d'un paquet de données, la plupart du temps due à la congestion du réseau.

Cet aspect de la QoS n'est pas l'objet de notre thèse, plusieurs travaux de recherche et ouvrages ont porté sur ce sujet [Chakraborty, et al.'03].

1.2.4. Niveau système

Au niveau système, on s'intéresse principalement à la gestion des ressources systèmes physiques et logicielles (Cf. Page.26) : on vérifie par exemple, la disponibilité des ressources physiques car souvent la dégradation de la QoS (e.g temps de réponse très élevé) est due à des ressources systèmes insuffisantes. L'intérêt du framework que nous proposons est d'aider à la mesure de l'utilisation des ressources systèmes et d'identifier celles qui causent les goulots d'étranglement. Car l'erreur à éviter est d'augmenter inutilement des ressources qui ne sont pas, à la base, responsables des dégradations des performances.

Autre aspect à considérer dans la gestion de la QoS est le choix du système d'exploitation. Une application n'offre pas les mêmes performances si elle est déployée sur un système Linux ou sur un système Windows car la gestion des ressources logicielles (Thread, processus, etc.) n'est pas du tout la même. En effet, Linux crée des « processus » plus rapidement que Windows 2000 par exemple, mais ordonnance « Schedule » moins vite les « threads » d'un même « processus ». Linux aura donc un fort avantage dans les cas où l'on doit créer et détruire beaucoup de threads ou processus, et ceci très fréquemment (typiquement un serveur web avec des scripts CGI créant un processus à chaque requête), mais sera contre performant dans un système où on utilise beaucoup de threads créés une fois pour toute (ex : base de données ou serveur web moderne ne créant pas de processus supplémentaire pour exécuter un script).

1.2.5. Niveau intergiciel « middleware »

Lorsque des services sont rajoutés au-dessus des bibliothèques systèmes, on les appelle usuellement des intergiciels (ou middleware), parce qu'ils servent d'intermédiaires entre les applications et le système d'exploitation. Il existe plusieurs types d'intergiciels:

- Intergiciels orientés composants .NET : ex. CLR (Common Library Runtime)
- Intergiciels orientés communication : ex. MOM (Message-Oriented Middleware) et CORBA.
- Intergiciels orientés composants Java : ex. les machines virtuelles Java (JVM)
- Etc.

De part leur emplacement stratégique, entre l'application et l'OS, l'étude des intergiciels constitue un des axes majeurs de la recherche. Dans l'objectif d'améliorer la QoS, certaines approches visent la construction d'intergiciels adaptatifs dotés de capacité de réflexion et de reconfiguration dynamique. Parmi ces travaux, nous citons le projet DREAM⁶, construit sur le modèle FRACTAL⁷, qui propose une bibliothèque de composants permettant de construire des middlewares de communication asynchrone dynamiques [Leclercq, et al.'04]. Cependant, ces approches restent encore, à notre connaissance, très peu employées en pratique de part leurs complexités mais continuent, toutefois, à alimenter la recherche du génie logiciel.

Actuellement, les approches courantes consistent à réaliser des réglages ou « tuning » des middlewares afin de trouver le bon paramétrage qui offre une QoS optimale. Certains middlewares fournissent des interfaces pour les interroger sur leurs contextes d'exécutions :

⁶ DREAM (Dynamic REflective Asynchronous Middleware) <http://dream.objectweb.org>

⁷ FRACTAL <http://fractal.objectweb.org>

c'est le cas, par exemple, de la machine virtuelle java (JVM) où l'interface JVMTI⁸ permet de profiler les applications Java et de surveiller l'activité du ramasse-miettes « garbage collector ».

1.2.6. Niveau applicatif

L'application peut être modifiée dans sa structure ou dans son fonctionnement afin d'améliorer la QoS fournie. La qualité de la conception, le choix de l'implémentation, la qualité du code ainsi que d'autres facettes de l'application sont des paramètres dont il faut tenir compte pour construire des applications performantes. Généralement, les contraintes de performance obligent à revoir les choix transactionnels effectués lors de la conception et à faire évoluer le code source de l'application. Au fil des années, le génie logiciel a proposé de nombreuses solutions technologiques dans le but de concevoir des applications performantes (patrons de conception⁹, conception par contrats, la programmation par composants, AOP¹⁰, etc.). Dans ce sens, nous soulignons les travaux de recherche dans le domaine de la programmation par composants qui visent à offrir aux composants logiciels le moyen de raisonner sur leurs conditions d'exécution à travers les contrats qu'ils souscrivent avec leur environnement de déploiement [Nicolas.'04]. Les composants auront la possibilité de modifier leurs conditions d'exécution en négociant dynamiquement les modalités de leurs contrats, et éventuellement de pouvoir adapter leurs comportements en fonction des contrats négociés. De telles applications sont capables de s'adapter elles-mêmes, de façon autonome aux évolutions de leur contexte d'exécution afin non seulement de continuer à fonctionner, mais aussi de tirer le meilleur parti des nouvelles possibilités qui peuvent apparaître dynamiquement [Kephart and Chess.'03].

1.3. Synthèse

Nous venons de présenter très rapidement certains aspects influant sur la QoS en général et sur les performances d'une application en particulier. Cette présentation, inévitablement incomplète et parcellaire, fait cependant ressortir l'importance de la structuration en couches des aspects abordés. A chaque couche, on peut identifier des facteurs qu'on peut ajuster pour atteindre un niveau de QoS satisfaisant pour l'utilisateur. Le Tableau 1 donne des exemples de facteurs qui influent sur la QoS au niveau de chaque couche.

⁸ Java Virtual Machine Tool Interface

⁹ « Design Patterns for Optimizing the Performance of J2EE Applications (<http://java.sun.com/developer/technicalArticles/J2EE/J2EEpatterns/>)

¹⁰ Aspect Oriented Programming (programmation par aspects) est un paradigme de programmation qui permet de réduire fortement les couplages entre les différents aspects techniques d'un logiciel (ex journalisation, sécurité, etc.). Cette technologie transverse n'est pas liée à un langage particulier (Java, C ou autre). La seule contrainte étant l'existence d'un tisseur d'aspect pour le langage cible

Couche	Exemples de facteurs
<i>Application</i>	<ul style="list-style-type: none"> ▪ Qualité de la conception ▪ Qualité de l'implémentation
<i>Intergiciels</i>	<ul style="list-style-type: none"> ▪ Paramétrage de la JVM (ex. Variable Xmx, Xms, etc.)
<i>Système</i>	<ul style="list-style-type: none"> ▪ Ressources physiques (ex. CPU, RAM, etc.) ▪ Ressources logicielles (ex. gestion des threads, gestion de la file d'attente, etc.)
<i>Réseau</i>	<ul style="list-style-type: none"> ▪ Capacité de la ligne ▪ Performances des équipements relais, etc.

Tableau 1: Synthèse des facteurs influant sur la QoS

Cette hiérarchisation plutôt guidée par la logique de l'évaluation de la QoS, s'articule sur trois notions de base:

- Le critère: c'est l'aspect de QoS (ou la propriété extra-fonctionnelle) concerné par l'évaluation. Dans notre cas, on s'intéresse particulièrement aux critères liés à la performance et qui seront abordés plus en détail dans le paragraphe suivant.
- Les facteurs: ce sont les paramètres qui influent sur les performances globales de l'application, certains de ces paramètres sont objectifs d'autres subjectifs.
- Les mesures: ce sont les métriques qui vont permettre de choisir, d'une manière quantitative, une configuration du système qui offre de meilleures performances.

De part la multitude des facteurs mis en jeu pour obtenir une QoS optimale, la gestion de celle-ci utilise un processus itératif basé sur le réglage ou « tuning » de ces facteurs. Cette opération requiert des outils de supervision capables de collecter des informations, de les analyser et de faciliter leur exploitation. Or, les outils actuels manquent généralement de généralité car ils ne permettent de superviser que des points spécifiques du système. D'où la nécessité de combiner plusieurs outils pour disposer de cette vue globale sur les performances des applications [Ivan, et al.'99].

Dans notre thèse, nous définissons l'architecture d'une plate-forme générique pour fédérer la supervision. Cette plate-forme aide à l'amélioration de la QoS des applications en fournissant un environnement homogène pour la tâche de collecte, d'analyse et de corrélation des mesures collectées sur les performances d'un système.

2. Concepts de l'évaluation des performances systèmes

Dans ce paragraphe nous nous intéressons au critère de la performance d'un système. Cet aspect de la QoS est important car il met en jeu plusieurs facteurs et métriques nécessaires pour son évaluation.

2.1. Définitions

Performance

Du point de vue de l'utilisateur, une application distribuée est performante si elle assure un temps d'exécution raisonnable suite à la sollicitation d'un client. Ce temps dépend d'une part des capacités de calcul et de communication des machines (sites du réseau) et d'autre part des canaux de communication logiciels (Intergiciels) et de la qualité de l'application (Cf. 1.2 Gestion de la QoS).

Ressource système

Une ressource système est une entité physique/matérielle (i.e., mémoire, processeur, disques, etc.) ou logicielle/conceptuelle (i.e., fichier, thread, file d'attente, etc.) qui participe à l'exécution d'une application et dont les services sont décrits par des caractéristiques quantitatives. L'étude des ressources systèmes utilisées par une application peut se faire à une granularité plus ou moins importante selon les besoins. Par exemple, l'étude de la mémoire consommée peut couvrir la mémoire vive réellement disponible, la mémoire mise en cache et la mémoire bufférisée. La commande « free -m » sous l'environnement Linux fournit ce genre d'information (Cf. Figure 2).

```
mhi@pcmhi:/opt/rt3$ free -m
```

	total	used	free	shared	buffers	cached
Mem:	885	604	280	0	192	115
-/+ buffers/cache:		297	588			
Swap:	478	0	478			

Figure 2 : Gestion de la mémoire sous Unix

2.2. Intérêts de l'évaluation des performances système

Les performances d'un système sont d'une importance capitale car leurs améliorations constituent bien souvent la motivation première lors de l'élaboration d'une nouvelle solution – ou d'une nouvelle implémentation. L'évaluation des performances consiste à étudier le comportement d'un système informatique en terme de ressources. Cette étude peut être utilisée pour déterminer la capacité de l'architecture d'un système informatique à supporter la charge et

pour identifier les parties de l'architecture où des problèmes de performance sont susceptibles de se produire. Elle peut être également employée pour comparer différentes architectures d'un système afin d'évaluer leurs mérites respectifs en terme de ressources nécessaires à la réalisation d'objectifs et d'exigences prédéterminées par les spécifications de l'entreprise.

2.3. Les techniques d'évaluation des performances

On peut citer quatre techniques différentes pour évaluer les performances d'un système [Jain.'91] :

- L'intuition : cette technique est dite également de raisonnement. Elle fait appel à un processus cognitif qui permet d'obtenir de nouveaux résultats ou de vérifier un fait en faisant appel à différentes « lois » ou « expériences ». En général, cette technique est suivie d'une étape de vérification pour prouver la validité du raisonnement.
- La mesure : cette technique a pour but de faire ressortir plusieurs indicateurs significatifs qui constitueront les « métriques de performance » [Katchabaw, et al.'96] pour quantifier les performances d'un système. En général, les deux métriques les plus utilisées sont les métriques de réponse (ex temps de réponse) et les métriques d'utilisation des ressources (ex ressources réseau ou ressources systèmes).
- La simulation : les modèles de simulation tentent, à partir de représentations distinctes de chaque composant d'un système et des interactions entre ces différents composants, d'en dériver le comportement du système global.
- Les modèles analytiques : ils modélisent le système à évaluer sous forme de variables et de paramètres en utilisant des techniques standards de modélisation comme les files d'attente [Kleinrock.'75] ou des réseaux de Pétri stochastiques[Marsan, et al.'95].

Avec l'augmentation de la complexité des systèmes informatiques actuels en terme de services opérationnels et de dépendances entre ces services, il est difficile d'évaluer les performances des systèmes par intuition comme il est difficile de construire des modèles analytiques afin de prévoir leurs performances. Les nouvelles approches du génie logiciel, basées sur les modèles tentent de répondre à cette problématique à travers la spécification des contrats au niveau des modèles [Collet and Rousseau.'05]. Cette technique, bien que prometteuse, reste difficile à mettre en œuvre puisqu'elle nécessite une maîtrise des propriétés fonctionnelles et extra-fonctionnelles avec une prise en compte en amont dès la phase de conception.

La technique par la mesure est la plus utilisée car ne nécessite aucune connaissance au préalable du système à évaluer. Nos travaux s'apparentent à cette technique et visent la construction d'une plate-forme pour fédérer les outils de mesures.

3. Besoins de l'industrie

Pour répondre aux exigences du marché en terme de fonctionnalités, de coût ou de flexibilité, les systèmes logiciels sont depuis longtemps poussés vers une modularité toujours accrue, qui passe notamment par la réutilisation de nombreux composants développés indépendamment (bibliothèques, intergiciels, etc.). Ce haut degré de réutilisation se traduit par des architectures toujours plus complexes, intégrant des éléments hétérogènes déployés sur des systèmes multi-plates-formes. Par conséquent, les systèmes de supervision doivent s'adapter à toutes ces contraintes.

De part son caractère industriel, le travail de cette thèse a été guidé par un cahier de charge qui décrit les spécifications et les caractéristiques d'une plate-forme fédératrice de la supervision. Nous avons retenu trois exigences que nous étudions à la lumière d'approches aussi bien académiques qu'industrielles. Ces exigences sont les suivantes :

- La performance ou la faible intrusivité de la plate-forme,
- La possibilité d'intégration des sondes hétérogènes,
- Le découplage entre la phase de collecte des informations et la phase de présentation.

3.1. Performance de la plate-forme de supervision

Nous définissons la performance d'un outil de supervision comme étant sa capacité à collecter et traiter des informations sans perturber ou modifier le comportement du système observé. En général, cette condition peut être atteinte de deux façons :

- En minimisant « l'intrusivité » des sondes,
- En réduisant la masse d'information que l'outil doit gérer tout au long du processus de supervision

3.1.1. Intrusivité des sondes

a. Définition

« L'intrusivité » dérive du mot intrusion qui veut dire « l'occurrence d'un processus ou d'un événement qui ne fait pas partie du contexte d'exécution d'une tâche mais qui interfère dans sa réalisation » [Ramchurn, et al.'04]. [Mansouri-Samani.'95] définit l'intrusivité d'un système de supervision par :

« Intrusiveness or the probe effect is the effect that monitoring may have on the behaviour of the monitored system, and results from the monitoring system sharing resources with the observed system (e.g., processing power, communication channels, storage space).....Monitoring may

affect the behaviour of the system under observation. This is particularly true when the application references external state or its behaviour is affected by race conditions. In general this intrusion or interference is undesirable and must be reduced or, when possible, completely eliminated. This can be achieved through suitable instrumentation techniques ».

Cette définition souligne que l'intrusivité est causée par l'interférence avec le système observé. Cette interférence peut s'opérer à deux niveaux : d'un côté le partage des ressources système (CPU, mémoire, etc.) et de l'autre côté le partage des ressources réseaux (bande passante). Ces deux facettes de l'intrusivité doivent être évaluées pour caractériser globalement l'intrusivité d'une sonde.

b. Intrusivité au niveau des ressources réseaux

[Goutelle and Primet.'03] définit globalement le degré d'intrusivité d'une technique de mesure comme étant la capacité à réaliser la mesure en générant le minimum de trafic supplémentaire. Cette définition, plutôt générale, met l'accent sur le volume du trafic généré par les sondes pour caractériser son intrusivité. On peut définir alors mathématiquement l'intrusivité par la formule donnée dans (Équation 1).

$$\text{Intrusivité} = \frac{C_x}{C_d} \%$$

Équation 1 : Calcul de l'intrusivité au niveau du réseau

C_x représente le débit du trafic supplémentaire généré par la sonde et C_d représente la bande passante. Par exemple, si on dispose d'une bande passante de 100 Mb/s et que l'activité de la supervision utilise un débit de 1Mb/s, alors l'intrusivité au niveau des ressources réseaux est de 1%.

Pour rappel, il existe deux techniques pour mesurer les caractéristiques d'un réseau (bande passante, délai, etc.) : *la métrologie active* dite également intrusive et *la métrologie passive* ou non-intrusive.

❖ **La métrologie active**: La méthode active consiste à injecter du trafic dans le réseau de manière contrôlée et à analyser les paquets retournés: taux de perte, délai, RTT¹¹, etc. Cette approche, qui permet d'étudier le comportement des composants et des protocoles sur le

¹¹ Round-Trip delay Time

trafic, possède l'avantage de prendre un positionnement orienté utilisateur puisqu'il lui permet de mesurer les paramètres du service dont il pourra bénéficier. En revanche, l'inconvénient majeur de cette approche est la perturbation introduite par le trafic de mesure qui peut faire évoluer l'état du réseau et ainsi fausser la mesure [Owezarski.'03]. Les mesures actives simples sont tout de même majoritaires dans l'Internet. Parmi eux, on peut citer les très célèbres commandes *ping* et *traceroute*.

- ❖ **La métrologie passive:** Le principe des mesures passives est de regarder le trafic et d'étudier ses propriétés en un ou plusieurs points du réseau. L'endroit idéal pour positionner des sondes de mesures passives est indéniablement dans les routeurs, car c'est par là que le trafic transite impérativement. L'avantage des mesures passives est qu'elles ne sont absolument pas intrusives, du point de vue réseau, et ne changent rien à l'état du réseau. En revanche, il est très difficile de déterminer le service qui pourra être offert à un client en fonction des informations obtenues en métrologie passive car ce type de métrologie ne permet pas de mesurer les limites des performances du réseau. Parmi les exemples d'outils utilisant la métrologie passive on peut citer le standard **SNMP**¹² et la commande « *ifstat* » pour l'environnement Linux.

c. Intrusivité au niveau des ressources système

L'intrusivité au niveau système existe lorsque les sondes d'observation requièrent des ressources systèmes qui pénalisent les performances de l'application observée. Cependant, il est difficile de trouver, dans la littérature, une définition qui nous convienne véritablement et qui traite ce type d'intrusivité sous ses différents aspects. Cette difficulté provient du fait qu'une ressource système peut avoir plusieurs niveaux de granularité. Si on considère l'exemple de la mémoire, on peut définir naïvement l'intrusivité au niveau de la mémoire comme étant le ratio entre la mémoire consommée par l'activité de supervision et la mémoire totale disponible. Etant donné que la mémoire existe sous plusieurs formes, on pourra alors définir l'intrusivité au niveau de la mémoire cache ou l'intrusivité au niveau de la mémoire physique, etc.

Par ailleurs, il est difficile de mesurer directement l'intrusivité d'un framework de supervision puisque l'outil de mesure a sa propre intrusivité. C'est pour cela, qu'en général, on raisonne en terme « *d'overhead* » où de surcharge engendrée par l'activité de supervision. Cette surcharge se traduit par une augmentation du temps d'exécution de l'application. Ainsi, on peut alors

¹² Simple Network Management Protocol

caractériser l'intrusivité d'une sonde, ou particulièrement d'une technique de mesure, par la différence des temps d'exécution avec ($Te_{avec\ supervision}$) et sans supervision ($Te_{sans\ supervision}$).

$$\begin{aligned} Overhead_{Temporel} &= Te_{avec\ supervision} - Te_{sans\ supervision} \\ Overhead_{\%} &= \frac{Te_{avec\ supervision} - Te_{sans\ supervision}}{Te_{sans\ supervision}} \times 100 \end{aligned}$$

Équation 2 : Calcul de l'overhead d'une activité de supervision

La mesure de l' « *overhead* » permet par exemple, de comparer des outils de supervision. Elle est utilisée également comme indicateur pour trouver un compromis entre la fréquence d'échantillonnage et le degré de perturbation de l'application supervisée.

d. Effets de l'intrusivité

En plus de la dégradation des performances du système observé, l'intrusivité peut causer:

- Le déséquilibrage des messages envoyés par les sondes de collectes
- L'incohérence des résultats obtenus
- L'augmentation du temps d'exécution du système
- Le phénomène du goulot d'étranglements « deadlock »
- La modification du comportement du système sous test

Le souci principal des plates-formes de supervision est de réduire cette intrusivité pour obtenir des mesures fiables. Pour cela, un grand intérêt est porté sur l'optimisation du fonctionnement des sondes de collecte. [Curtis and David.'02] décrivent, par exemple, plusieurs optimisations permettant de collecter de façon efficace les informations relatives à la consommation des ressources système, pour un environnement Linux, et ceci en utilisant le système de fichier **/proc**.

En plus de l'intrusivité, la masse d'information générée et gérée par les plates-formes de supervision peut être un facteur déterminant de leurs performances. Dans la suite du rapport, nous étudions ce critère de performance bien qu'il est parfois difficile, comme nous le verrons par la suite, de le dissocier du critère d'intrusivité vu qu'une masse importante d'information peut souvent être la cause d'une intrusivité que ce soit du point de vue système ou réseau.

3.1.2. Masse d'information

La masse d'information générée par les sondes influe sur les performances d'une plate-forme de supervision de deux façons. D'une part, la collecte d'une masse importante d'information engendre un surcoût en terme d'utilisation des ressources systèmes et d'autre part, si les informations sont transmises au fur et à mesure de leur collecte, cela risque de générer du trafic sur le réseau et peut perturber le fonctionnement de l'application.

Pour augmenter les performances d'un framework de supervision, [Curtis and David.'02] proposent deux solutions : d'abord restreindre la masse d'information à celles qui sont pertinentes, ensuite transmettre ces informations moins fréquemment sur le réseau.

Pour réduire la masse d'information, les mêmes auteurs proposent deux techniques:

- Ajuster la fréquence de collecte
- Filtrer les informations

a. Ajuster la fréquence de collecte

La fréquence de collecte correspond au nombre de mesures effectuées par unité de temps. Réduire cette fréquence permet de réduire la masse d'information. Néanmoins, cette solution doit être mise en place en cohérence avec les objectifs de la supervision. Par exemple, dans l'objectif de faire du profiling¹³, il est nécessaire de choisir des fréquences élevées car la courbe d'utilisation des ressources systèmes peut varier très fréquemment. Pour d'autres objectifs de supervision, on peut envisager des fréquences de collecte relativement faibles. C'est le cas en général, pour le « monitoring » en temps réel à l'aide d'outils tels que TPTP¹⁴, NAGIOS¹⁵ ou SNMP. Ce qui importe pour ce type de supervision c'est d'avoir des graphes de tendance qui seront très utiles pour anticiper les évolutions des systèmes supervisés.

b. Filtrer les informations

Les filtres limitent les informations collectées par les sondes en rejetant celles qui ne sont pas nécessaires. La conception des filtres et leurs emplacements ont un impact majeur sur l'efficacité d'un filtrage. En effet, [Al-Shaer.'96] souligne que les algorithmes de filtrage et les structures de données utilisées dans les filtres déterminent le nombre de comparaisons nécessaires pour classer une information. Cette opération peut être coûteuse en terme de

¹³ Consiste à obtenir un certain nombre de mesures sur l'exécution d'un programme : nombre de passages dans une fonction ou dans certaines structures de contrôle (conditionnelles, boucles, etc.)

¹⁴ <http://www.eclipse.org/tptp>

¹⁵ <http://www.nagios.org>

consommation des ressources systèmes, d'où l'intérêt de ne pas réaliser le filtrage sur la machine où est déployée l'application supervisée.

Les emplacements des filtres sont de trois niveaux comme l'illustre la Figure 3.

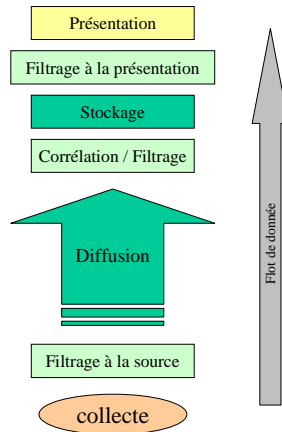


Figure 3 : Circuit de traitement des données de supervision

Filtrage à la source

Ce filtrage est réalisé sur la machine où se trouvent les sondes et permet de réduire les informations transmises sur le réseau. La structure du filtrage ne doit pas être complexe pour ne pas engendrer de surcoût en terme d'utilisation des ressources de la machine. Un des exemples d'application du filtrage au plus près des sources est le domaine de la détection d'intrusion (ex SNORT¹⁶), où les sondes disposent d'un ensemble de filtres (appelés également signatures) qui décrivent les paquets qui correspondent à des scénarios d'attaques.

Corrélation/Filtrage

La corrélation consiste à croiser les informations en provenance d'une ou plusieurs sources afin de disposer d'une vue globale du comportement d'une application. Par exemple, pour tester les performances d'une application web, on cherche à corréler la charge du trafic injecté avec la consommation des ressources systèmes. On cherche également à corréler le nombre des utilisateurs avec le temps de traitements des requêtes.

En corrélant les mesures de performances, ainsi que d'éventuelles informations additionnelles (ex. des fichiers journaux), on peut espérer réduire le volume d'information à traiter, améliorer la qualité du diagnostic proposé et dégager une meilleure vision globale de l'état du système supervisé.

¹⁶ SNORT est un système de détection d'intrusion (IDS Intrusion Detection System) Open Source (<http://www.snort.org>)

La tâche de corrélation peut être facilitée par l'utilisation d'un même formalisme pour encoder les informations collectées. Généralement, les approches basées sur le formalisme XML (eXtensible Markup Language) sont les plus utilisées pour pouvoir gérer l'hétérogénéité des informations (Cf. Paragraphe : Hétérogénéité des formats des données, page.37)

Filtrage à la présentation

Ce type de filtrage permet à l'utilisateur d'afficher uniquement les informations qui l'intéressent parmi celles stockées. En pratique, l'utilisateur dispose d'un formulaire pour sélectionner les types d'informations et les critères associés. Implicitement, cela déclenche une suite de requêtes vers le médium de stockage (ex système de base de données) qui vérifie la disponibilité des informations demandées et qui renvoie le résultat.

3.1.3. Synthèse

Mesurer les performances d'une application est essentiel pour mieux comprendre sa dynamique et anticiper les évolutions et les optimisations. Cependant, la réalisation de cette activité nécessite des outils utilisant des techniques de collecte qui soient peu-intrusives. Cette intrusivité peut être coûteuse en terme de consommation des ressources systèmes et peut perturber le comportement de l'application ou du système supervisé. De plus, la réduction de la masse d'information peut s'avérer très utile pour augmenter l'efficacité de l'activité de la supervision et faciliter le diagnostic. Le filtrage et l'ajustement de la fréquence de collecte sont des mécanismes parmi d'autres qui permettent cette réduction.

Outre l'aspect intrusivité, la possibilité de fournir un environnement commun où des sondes hétérogènes peuvent fonctionner ensemble nous semble très utile pour la supervision. Dans la suite, nous exposons l'intérêt de cette caractéristique et nous présentons quelques techniques qui facilitent cette intégration.

3.2. L'intégration des sondes hétérogènes

3.2.1. Introduction

Les systèmes actuels sont construits par intégration de composants existants. Il en résulte des architectures complexes (hétérogènes, distribuées, etc.). D'une part, la contrainte d'hétérogénéité peut prendre plusieurs formes : hétérogénéité de l'environnement d'exécution (Linux, Windows, etc.) et hétérogénéité des formats de données manipulées (XML, formats non standardisés, etc.). La supervision de tels systèmes nécessite parfois de combiner plusieurs outils de mesure. Cela pose cependant un problème d'interopérabilité entre les résultats générés par ces outils; ces résultats peuvent être générés dans des formats différents (XML, CSV, etc.) et nécessitent d'abord d'être uniformisés avant de les exploiter. D'autre part, mesurer les

performances d'un système distribué nécessite de placer des sondes sur les différentes parties de ce système. L'horodatage des informations collectées est indispensable à la supervision parce qu'il permet de savoir à quel moment la mesure a été effectuée. Le problème qui peut se poser à ce stade est celui de la synchronisation des informations.

Dans cette partie, nous dressons un état de l'art des techniques utilisées pour uniformiser les formats de données et celles utilisées pour les synchroniser.

3.2.2. Contrainte d'hétérogénéité

Mesurer les performances d'un système construit à partir de modules hétérogènes n'est pas une tâche facile. Il implique, à la fois, des hétérogénéités au niveau des environnements d'exécutions (Linux, Windows, etc.) et au niveau des formats de données manipulées par le système (XML, CSV, etc.).

a. Hétérogénéité des environnements de déploiement

Exemple illustratif

La Figure 4 présente une architecture web composée de trois parties qui sont : le(s) serveur(s) web « web server », le proxy et un système de base de données (SGBD).

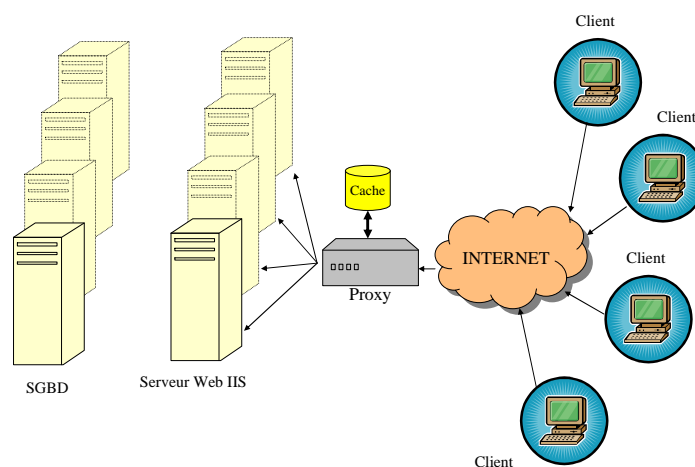


Figure 4: Architecture classique d'une application web

Cet exemple a le mérite d'illustrer la contrainte d'hétérogénéité des environnements de déploiement vu que le *proxy* et le serveur web sont déployés, généralement, sur deux systèmes d'exploitation différents. De plus, il s'agit de l'une des configurations les plus classiques et les plus courantes pour améliorer les performances d'une application web. En effet, le *proxy* permet d'améliorer les performances, en réduisant le trafic réseau et en réduisant la charge des serveurs. Dès qu'un document est sauvegardé dans le cache du « proxy », plusieurs requêtes Web peuvent

être satisfaites directement à partir de ce cache, sans générer du trafic supplémentaire depuis et vers les serveurs. Cet exemple d'architecture peut être enrichi en supposant l'utilisation de plusieurs serveurs web et/ou plusieurs proxy afin d'assurer un équilibrage de la charge.

Les cas d'études

Pour l'exemple ci-dessus, plusieurs cas d'études peuvent être envisagés pour illustrer l'intérêt de la corrélation :

- Le dimensionnement : dimensionner les ressources systèmes sur chaque nœud pour obtenir un temps de réponse moyen correct pour l'affichage d'une page. Ce cas par exemple nécessite une corrélation entre les informations liées à l'utilisation des ressources systèmes.
- La détection des goulots d'étranglement : il s'agit de trouver les points « chauds » de l'architecture (proxy, serveur web, SGBD¹⁷, etc.) et les ressources insuffisantes (processeur, mémoire, bande passante, etc.) pour anticiper les évolutions. Les informations à corréler sont les mêmes que celles du cas précédent.
- Les tests aux limites : il s'agit de vérifier, pour une architecture donnée, le nombre maximal d'utilisateurs qui peuvent être servis simultanément tout en garantissant un temps de réponse raisonnable pour l'utilisateur. Pour ce cas, on cherche particulièrement à corréler le temps de réponse avec le nombre d'utilisateurs.

Comment mesurer les performances systèmes ?

Pour mesurer la consommation des ressources système, il existe des standards tel que SNMP [RFC.1157.], qui offre la généricité souhaitée et qui permet de collecter des informations aussi bien dans un environnement Unix que Windows [Subramanyan, et al.'00]. D'autres alternatives à SNMP consistent à utiliser des sondes spécifiques à chaque environnement. Les techniques utilisées par ces sondes pour collecter ces informations sont diverses :

- Pour l'environnement Linux, comme c'est le cas pour les autres systèmes d'exploitation Unix, l'état du noyau en général et celle des ressources systèmes en particulier, sont réifiés à travers un système de fichiers virtuel, généralement monté sous l'entrée du système de fichiers /proc. Reste la manière d'accéder à ces fichiers (ligne par ligne, lecture en bloc, etc.) et la façon de les gérer. Dans ce sens, [Curtis and David.'02] décrit plusieurs optimisations permettant de collecter de façon efficace des données depuis ce système de fichiers.

¹⁷ Système de Gestion des Bases de Données

- Pour l'environnement Windows, il existe une infrastructure d'observation qui permet d'accéder aux informations des performances systèmes. [Knop, et al.'02] décrit comment collecter ces informations et propose l'outil **WatchTower** basé sur l'interface PDH (Performance Data Helper), qui masque la complexité de la base de registre de Windows.

Comment mesurer les temps de réponses ?

Généralement, les outils de test en charge fournissent ce genre de mesures et permettent en plus de générer des flux internet en simulant le comportement des utilisateurs. *OpenSTA* est l'une référence du domaine; les informations qu'il fournit sont exportables dans des fichiers au format CSV¹⁸.

Le défi à relever

Le grand défi dans l'exemple que nous citons, est de pouvoir analyser et exploiter dans le même environnement aussi bien les mesures de performance collectées par les sondes systèmes Linux/Windows que celles fournis par *OpenSTA*.

b. Hétérogénéité des formats des données

Disposer d'une vue globale du comportement d'une application nécessite, en général, la corrélation de plusieurs sources d'informations (information de performance, informations applicatives, etc.). Ces informations sont fournies par des outils différents et qui, en plus, gèrent des formats de données différents. La Figure 5 donne l'exemple de deux fichiers hétérogènes où le premier correspond à un fichier généré par l'outil de test en charge *OpenSTA* et le deuxième est un fichier généré par la sonde *Vmstat* que nous décrivons plus loin dans ce manuscrit.

```
===== Extrait du fichier VUsers.txt de l'outil OpenSTA =====  
  
1132674056,0,0,  
1132674058,1,1,  
1132674060,2,2,  
1132674062,2,2,  
.....  
-----  
  
===== Extrait du fichier généré avec la sonde Vmstat =====  
  
11/22/2005 16:39:13,1,127104  
-PROCS_R PROCS_B MEM_SWAP MEM_FREE MEM_BUF MEM_CACH SWAP_SI SWAP_SO S CPU_USR CPU_SYS CPU_IDL CPU_WAIT  
2 1 5636 61588 2632 28200 8 6 1 8 6 0 0 0 98 1  
0 0 5636 61876 2632 28204 0 0 0 0 1010 18 0 1 99 0  
0 0 5636 61876 2632 28204 0 0 0 0 1002 11 1 0 99 0  
....
```

Figure 5 : Extrait des fichiers générés respectivement par *OpenSTA* et *Vmstat*

¹⁸ CSV Comma-Separated Values

Le fichier VUsers.txt décrit l'évolution du nombre d'utilisateurs virtuels dans le temps. Le premier champ est le champ d'horodatage et le deuxième et troisième champ correspondent respectivement au nombre d'utilisateurs inactifs et actifs.

Le fichier généré par la sonde Vmstat comporte les informations relatives à l'utilisation des ressources systèmes. La première ligne contient la date d'exécution de la sonde et la fréquence de la collecte, la deuxième ligne contient les intitulées des informations collectées et les lignes qui suivent contiennent les informations collectées.

Cet exemple est récurrent dans le domaine du test en charge où il est souvent question de trouver des solutions qui facilitent la corrélation entre les résultats de test et les performances des systèmes testés.

Pratiques actuelles

Les pratiques courantes pour gérer le problème d'hétérogénéité des informations de supervision sont basées généralement sur l'utilisation des tableurs (ex Macros Excel) ou sur l'utilisation d'un système de base de données. La première solution est difficile à envisager pour des grandes quantités d'information. Quant aux systèmes de base de données, l'inconvénient majeur est qu'ils génèrent un trafic réseau qui peut éventuellement le surcharger et créer, par conséquence, un effet de gigue.

Autre contrainte liée à l'exploitation des informations est illustré dans la Figure 6.

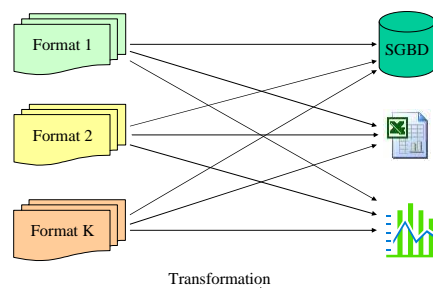


Figure 6: Export des fichiers hétérogènes

Si on dispose de K fichiers qui sont hétérogènes et que l'on cherche à les exporter vers N formats différents, il faut alors développer (N x K) transformations. Ceci apparaît contraignant si on doit gérer plusieurs formats de données en entrées. Cette problématique n'est pas nouvelle, elle a été en particulier traitée dans le domaine de l'analyse des fichiers journaux (logs) où le problème d'hétérogénéité est plus abondant. Il a fait l'objet de plusieurs propositions de

standardisation dont le format ULM¹⁹ [Debeaupuis and Abela.]. L'idée consiste à proposer un formalisme d'encodage pivot pour faciliter l'analyse et la corrélation des fichiers journaux (Cf. Figure 7).

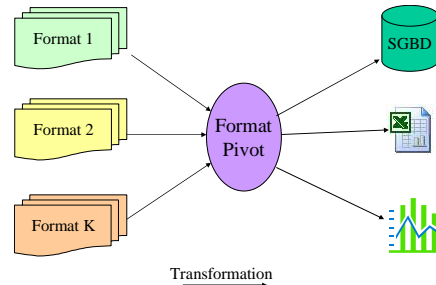


Figure 7 : Utilisation d'un format pivot pour la transformation

Pour illustrer la syntaxe ULM, nous prenons l'exemple d'une entrée du fichier système **Syslog**²⁰. Nous donnons la syntaxe associée dans le langage ULM et dans un autre langage basé sur XML (Cf. Figure 8)

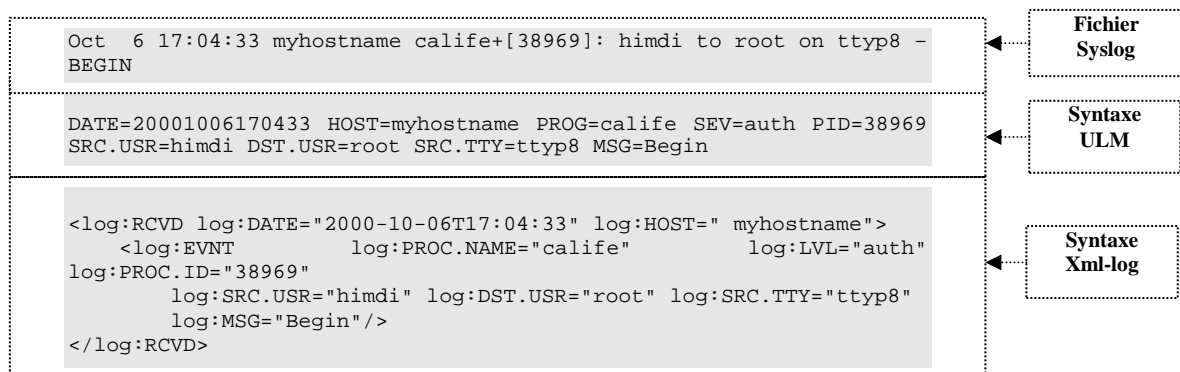


Figure 8 : Exemples de la syntaxe ULM et Xml-log

La syntaxe d'ULM est structurée sous la forme du couple {clé=valeur} ou les clés correspondent à des attributs prédéfinis (ex DATE, PROG, MSG, etc.) et au nombre limité. Cette contrainte de limitation du nombre d'attributs a conduit le groupe Syslog-Sec de l'IETF à définir une syntaxe basée sur le formalisme XML. C'est ainsi que les spécifications Xml-Log, une « XML-isation » d'ULM, ont été standardisées en 1999.

¹⁹ Uniform Logger Messages

²⁰ Le fichier Syslog contient les messages envoyés par le système (messages d'erreurs, des avertissements et d'autres messages)

Actuellement, les formalismes d'encodages sont dans leur majorité basés sur l'encodage XML. Cela s'explique par la richesse et l'extensibilité du formalisme et également par l'existence d'outils qui facilitent l'analyse de données XML. Partant de ce constat, notre choix s'est porté sur le format CBE²¹ qui est un formalisme soutenu par le consortium OASIS²² et dont il existe un moteur générique d'analyse, développé en java, qui permet de transformer des informations hétérogènes vers le format CBE.

3.2.3. Synthèse

Pour disposer d'une vue globale des performances d'une architecture logicielle, il est parfois nécessaire de combiner plusieurs solutions hétérogènes adaptées pour collecter des informations spécifiques sur cette architecture. Par conséquent, le principal défi pour un framework de supervision est celui de fournir un environnement où ces solutions hétérogènes peuvent fonctionner ensemble.

C'est l'idée que nous défendons dans cette thèse à travers l'intégration d'un module générique d'adaptation qui transforme les informations hétérogènes dans le format standard CBE. Ce même module a été évolué pour permettre l'application de traitements supplémentaires sur les informations ce qui facilite leurs exploitations directes.

Un dernier besoin qui a son importance dans la supervision concerne le découplage entre les phases de collecte et de présentation des informations. Le paragraphe suivant illustre par des exemples, l'intérêt de ce découplage car il offre la possibilité d'insérer, d'une manière flexible, des traitements dans le processus de supervision.

3.3. Découplage entre la collecte et la présentation

3.3.1. Introduction

D'une manière générale, un framework de supervision distribuée dispose de quatre principaux modules légèrement connectés entre eux [Mansouri-Samani.'95] (Cf. Figure 9).

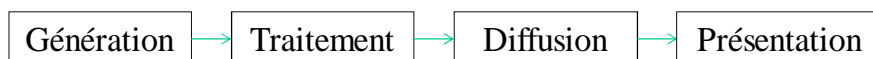


Figure 9: Principaux activités de la supervision

- La génération ou l'acquisition correspond à l'activité de collecte des informations,

²¹ Common Based Event

²² Organization for the Advancement of Structured Information Standards

- Le module traitement (ou « processing ») fournit un ensemble de fonctionnalités telles que la corrélation, le filtrage et le formatage des informations de supervision. Ces informations sont transformées, à l'issue de ces fonctionnalités, dans le format requis avec le niveau de détail souhaité,
- L'activité de diffusion (ou « dissemination » en Anglais) consiste à transmettre les informations à d'autres modules en vue d'appliquer des traitements additionnels (ex stockage, etc.),
- L'activité finale de la supervision consiste à présenter les informations collectées et traitées à l'utilisateur final.

Entre la génération et la présentation des informations, on peut multiplier et intervertir l'ordre des deux activités de traitement et de diffusion. C'est généralement l'outil de supervision qui impose le « workflow » des informations.

Concernant la caractéristique du découplage dont on parle ici, cela consiste à avoir une phase de traitement complètement dé-corrélée de la phase de collecte et celle de présentation. Les informations sont d'abord collectées puis traitées avant d'être présentées.

3.3.2. Intérêt du découplage

Dans le domaine de la supervision, on distingue deux grandes familles : la supervision « *online* » et la supervision « *offline* ». La première, dite aussi la supervision en temps-réel ou « on-line monitoring », consiste à collecter, analyser et présenter les informations de supervision au cours de l'exécution de l'application. Dans le cas contraire, la supervision est dite « *offline* ».

Le grand avantage de la supervision « *online* » est celui de rendre accessible les mesures collectées en temps réel. En revanche, cela se fait au détriment des performances à cause de l'intrusivité engendrée par le traitement et l'analyse des informations au fur et à mesure de leur collecte [Mansouri-Samani.'95]. D'où l'intérêt porté sur l'approche « *offline* » basée sur une analyse postérieure²³ des informations. Parmi les exemples des outils de supervision « *online* », nous citons la plate-forme TPTP et NAGIOS (Cf. pages 50 et 53).

L'approche « *offline* » est plus flexible en terme de traitement car les contraintes temporelles n'existent pas. Cette flexibilité nous sera utile car il va nous permettre de synchroniser les informations par décalage des «offset» temporelles. La synchronisation est sans doute l'une des grandes difficultés qui peuvent être rencontrées dans le cadre d'une supervision distribuée ou

²³ où post-analysis

d'une manière générale, dans un contexte des machines parallèles. De nombreux articles traitent ce sujet dont l'article de J.M JEZEQUEL [Jezequel.'89]. D'un point de vue pragmatique, nous distinguons entre trois formes de synchronisation : en amont, en cours ou après la phase de collecte. Dans la suite, nous exposons les techniques mises en œuvre pour réaliser ces différentes formes de synchronisation ainsi que les avantages/inconvénients de chacune.

a. Synchronisation en amont

Il s'agit d'une synchronisation au niveau des horloges des machines sur lesquelles sont déployées les sondes de supervision. Les horloges sont synchronisées à l'aide d'un protocole tel que NTP²⁴ (Cf. ANNEXE A): Sur les machines, on installe des clients NTP qui se charge du réglage automatique des horloges. Il existe plusieurs implémentations des clients NTP aussi bien pour l'environnement Linux que Windows. On peut utiliser, par exemple, la commande *ntpdate* pour synchroniser une machine linux avec un serveur de temps distant.

NTP reste un des protocoles les plus utilisés pour assurer la synchronisation des machines dans un réseau. La précision obtenue avec ce mécanisme est de l'ordre de quelques millisecondes ce qui suffit largement pour de nombreux contextes de supervision comme c'est le cas de la mesure des performances des systèmes.

b. Synchronisation en cours de supervision

Il s'agit d'une synchronisation au fur et à mesure de la collecte des informations. Le principe de cette technique consiste à ré-estampiller les informations collectées à l'aide d'un ou plusieurs médiums ou concentrateurs synchronisés entre eux comme l'illustre la Figure 10.

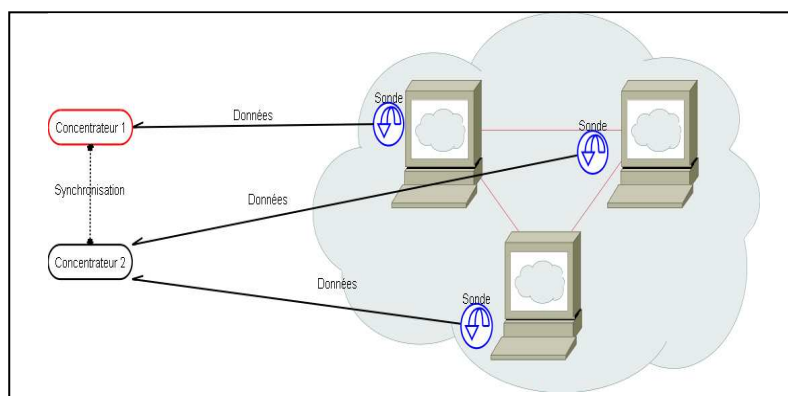


Figure 10 : Synchronisation par un médium

²⁴ Network Time Protocol

Cette technique constitue une alternative pour assurer une synchronisation en temps réel des informations collectées. Parmi ses applications, on trouve généralement les plates-formes de supervision utilisant une base de données (ex CACTI²⁵).

Pour assurer la fonction de datation, le concentrateur doit disposer d'un mécanisme intrinsèque d'horodatage qui estampille automatiquement l'information à la réception. C'est ce qu'on trouve généralement dans les bases de données (ex MySQL²⁶) et les middlewares de communication (ex JORAM²⁷). La précision de ce mécanisme dépend en premier lieu des délais de transmission sur le réseau : plus grande est la variation de ces délais moins grande est la précision de cette technique (Cf. Figure 11)

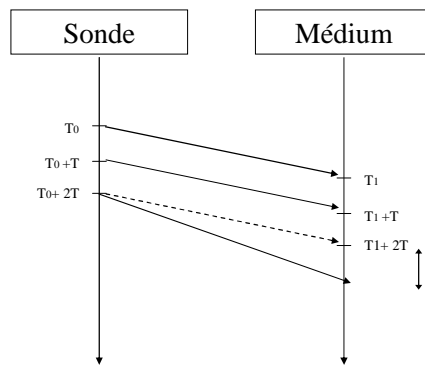


Figure 11 : Précision d'une synchronisation par un médium

La valeur de δ représente la gigue et varie en fonction de l'état du réseau. Généralement, le médium est déployé sur une machine différente de celles où sont déployées les sondes pour ne pas surcharger le système supervisé.

Vu son mode de fonctionnement, ce type de synchronisation présente une autre contrainte : toutes les informations collectées doivent transiter impérativement par un des médiums. Cette contrainte est génératrice de trafic sur le réseau ce qui risque de le perturber et de créer des variations dans les délais de transmissions (gigue). D'autre part, les médiums doivent être bien dimensionnés pour gérer la masse d'information et éviter d'être le goulot d'étranglement de l'application.

²⁵ <http://cacti.net/>

²⁶ <http://www.mysql.com/>

²⁷ <http://joram.objectweb.org/>

c. Notre approche : une synchronisation en aval

Nous avons vu qu'avec une synchronisation en cours de supervision, le dimensionnement des médiums et la stabilité du réseau sont les deux points chauds de ce mécanisme, d'autant plus, que cette technique n'est pas applicable dans des contextes où les informations sont déjà stockées dans des fichiers. D'où l'intérêt d'une approche basée sur la synchronisation en aval qui consiste à recalculer les échelles temporelles par rapport à un même référentiel de temps. Cette technique nécessite de calculer les décalages « offset » entre les horloges des sources (i.e. horloges des sites où les informations ont été collectées) et l'horloge de référence (Figure 12).

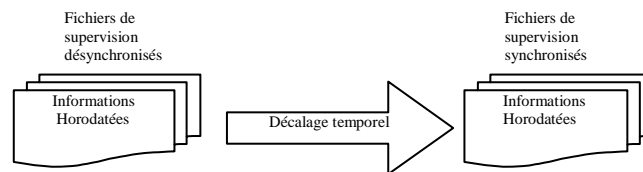


Figure 12 : Synchronisation en aval

Contrairement au deuxième mode de synchronisation, cette approche ne génère pas de trafic durant la phase de collecte car les informations sont enregistrées localement. Il ne dépend pas de la charge de réseau car la mesure des « offset » peut être réalisée lorsque la supervision prend fin. Toutefois, l'applicabilité de l'approche est liée à deux conditions : d'abord au découplage entre la phase de collecte et la phase de présentation ce qui permet d'insérer des traitements personnalisés, ensuite à l'absence du phénomène du *dérive des horloges physiques*. Cette dernière condition est aussi valable pour la deuxième approche de synchronisation. Dans cette thèse, nous considérons que la dérive d'horloge est négligeable sur les échelles de temps durant lesquelles les phases de supervision ont lieu. Cette approximation est justifiée du fait que la précision des mesures qui nous intéressent est de l'ordre de la seconde (ou du dixième de seconde), et qu'une marge d'erreur de 10^{-1} s est acceptable. La durée de supervision étant de quelques heures, les horloges dérivant en moyenne de 10^{-6} , le décalage est inférieur à 10^{-2} secondes. Toutefois, si un besoin de précision supérieur était nécessaire, l'approche proposée dans cette thèse resterait valable en appliquant des algorithmes de propagation et de synchronisation d'horloges distribuées comme ceux proposés dans [Jezequel.'89]

4. Conclusion

Dans un souci d'offrir une meilleure Qualité de Service au client, l'évaluation des performances des systèmes est d'une importance capitale. Cette évaluation permet de mieux connaître la dynamique du système évalué et d'anticiper son évolution future (ex augmentation des ressources physiques).

Pour qu'elle soit objective, l'évaluation des performances doit se baser sur des propriétés observables qu'on appelle « métrique de performances ». Les principales métriques sont le temps de réponse et la consommation des ressources systèmes et réseaux. Ces métriques peuvent être mesurées à l'aide d'outils intégrés, mais souvent les développeurs ont recours à des solutions *ad hoc* en combinant des solutions jugées meilleures que d'autres. Or, la difficulté pour cette approche est l'absence d'une plate-forme qui permet à ces solutions de fonctionner ensemble. **C'est la motivation principale de cette thèse : construire une plate-forme générique pour fédérer des outils de supervision hétérogènes. Cette plate-forme vise à faciliter la mesure des performances des systèmes distribués.**

Notre démarche, pour construire cette plate-forme, a été guidée par l'étude de trois exigences principales qui constituent les principaux besoins de l'industrie en matière de supervision: une solution peu-intrusive, la possibilité d'intégrer des sondes hétérogènes et le découplage entre l'acquisition et la présentation des mesures de performance. Ces caractéristiques (où exigences) nous semblent fondamentales puisqu'elles permettent respectivement d'obtenir des résultats fiables, complets et synchronisés.

Dans le second chapitre de cette partie, nous dressons un état de l'art des différentes solutions pour mesurer les performances des systèmes. Nous décrivons les architectures de ces solutions et nous démontrons leurs limites par rapport aux exigences citées précédemment.

C. Outils et techniques de mesure des performances systèmes

Il existe plusieurs architectures et techniques pour mesurer les performances des systèmes. Ces solutions vont de petits scripts jusqu'à des architectures complètes de supervision. Ce chapitre a pour but de montrer le contexte technologique de la thèse, et en particulier les principaux standards et frameworks de supervision.

Dans cet état de l'art, nous présentons le standard SNMP¹, intégré dans plusieurs outils de supervision (ex HP OpenView², NAGIOS³, etc.) et dont l'étude va permettre de déduire les contraintes liées à son utilisation.

Nous présentons également la plate-forme TPTP⁴, un projet Eclipse avec diverses fonctionnalités dont un moteur générique d'analyse qui permet la mise en forme des informations hétérogènes.

Nous présentons aussi, l'outil NAGIOS qui a le mérite d'être l'une des références open source dans le monde de supervision temps-réel. Son principal atout est la facilité de développer des sondes pour collecter des informations sur le système.

Pour finir, nous citons un projet issu de la recherche académique sur les modèles de composants. Il s'agit du framework LeWYS, une bibliothèque de sondes basées sur le modèle FRACTAL. Bien que le projet ne soit plus maintenu par la communauté, il a le mérite d'être cité dans notre rapport de part son concept novateur qui vise à construire des sondes génériques avec des capacités de reconfiguration dynamique.

¹ Simple Network Management Protocol

² <http://h20229.www2.hp.com>

³ <http://www.nagios.org>

⁴ Test & Performance Tolls Platform

1. Standard SNMP

1.1. Principe

SNMP est un protocole d'administration de réseau qui se veut simple dans son utilisation. Son concept lui permet de gérer aussi bien des informations applicatives que des informations liées à l'utilisation des ressources systèmes (processeur, mémoire, charge des interfaces réseaux, etc.). Afin de pouvoir l'utiliser, il est nécessaire que les éléments à superviser soient équipés d'un agent SNMP. L'information qu'on pourra recueillir dans un agent SNMP se trouve regroupée dans la MIB⁵ que nous présentons dans le paragraphe suivant. Ce protocole étant basé sur un modèle Client/Serveur (Cf. Figure 13), la station d'administration émet un paquet UDP⁶ représentant une requête SNMP à destination de l'équipement à superviser. L'agent, qui met à jour régulièrement sa table MIB, retourne dans un paquet UDP vers la station d'administration, les valeurs des variables SNMP demandées.

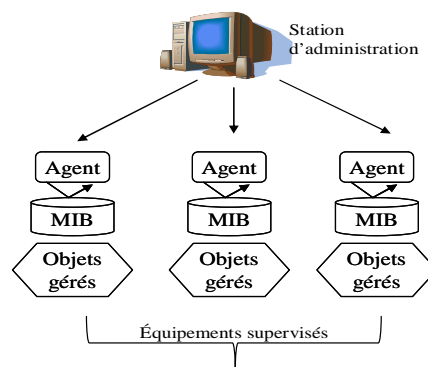


Figure 13 : Architecture du SNMP

1.2. MIB

Chaque agent SNMP met à la disposition de la station d'administration des objets qui sont structurés dans un MIB (Management Information Base) définie par la RFC-1213. Chaque objet est identifié dans la MIB par un *OID* (Object Identifier) qui définit sa position au sein de la structure arborescente de la MIB (Figure 14).

⁵ Management Information Base

⁶ User Datagram Protocol

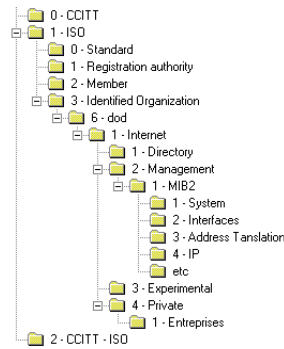


Figure 14 : Objets de la MIB

De la même façon, les informations relatives à la consommation des ressources systèmes sont accessibles par leur OID. Sur le site « **mibDepot**⁷ » on peut consulter les Oid de plusieurs équipements réseaux et systèmes d'exploitation. Nous exposons dans ANNEXE B Quelques Oid pour mesurer les performances systèmes d'un environnement Linux.

1.3. Atouts et faiblesses du standard SNMP

SNMP fonctionne suivant un mode synchrone (client/serveur), cela veut dire que le client doit solliciter, à chaque fois, l'agent pour connaître la valeur actuelle d'une métrique de performance. Ce mode de fonctionnement est l'une des faiblesses de SNMP et cette faiblesse est partagée avec d'autres outils de supervision fonctionnant de la même façon (ex outil NAGIOS).

De nombreux travaux de recherche traitent la question de l'intrusivité du protocole SNMP. Dans [Pattinson.'01], **Pattinson** étudie ce point en terme de trafic (requêtes/réponses) généré respectivement par la station d'administration et par les agents SNMP. Les facteurs pénalisants sur l'intrusivité de SNMP sont:

- La fréquence de collecte,
- Le nombre d'équipements supervisés.
- Le nombre d'objets (OID) collectés
- Le type des objets collectés (String, Integer, Ip Adresse, etc.)

Dans un autre article, [Pras, et al.'04] comparent les performances des agents SNMP avec plusieurs prototypes d'agents utilisant la technologie des web services. Les aspects : consommation des ressources (systèmes et réseau) et temps de réponse constituent les critères de cette comparaison. Les expériences montrent que SNMP consomme moins de ressources réseau si un seul objet est collecté. Selon les auteurs, cela s'explique par la différence des

⁷ <http://www.mibdepot.com>

formalismes d'encodages utilisés : d'un côté BER⁸ pour SNMP et de l'autre côté XML pour les web services. Il faut noter que le codage BER est gourmand en octet car un paramètre en un octet sera codé sur trois octets (l'entier 12 est encodé comme \$02 \$01 \$0C).

1.4. Discussion

Le protocole SNMP, comme l'acronyme l'indique, est simple et c'est d'ailleurs l'une des raisons de sa popularité. De plus, il est adopté par de nombreux constructeurs des équipements réseaux comme l'outil incontournable pour superviser leurs équipements. Cependant, le mode de fonctionnement du protocole basé sur une communication synchrone (requête/réponse) est pénalisant en terme d'intrusivité au niveau du réseau car il est générateur de trafic. De plus, la fréquence de collecte doit être choisie en fonction du nombre des équipements supervisés ce qui constitue une limite pour un large passage à l'échelle.

Le standard SNMP est utilisé dans le cadre d'une supervision « *online* » et donc hérite des avantages et des inconvénients de ce type de supervision. La synchronisation est réalisée au niveau de la machine d'administration qui constitue d'ailleurs le point « chaud » de l'architecture d'une plate-forme de supervision utilisant SNMP.

2. Eclipse TPTP

2.1. Introduction

Eclipse⁹ est un environnement de développement intégré (IDE), ouvert et open source. Le projet a été lancé par IBM et largement utilisé grâce à son architecture basée sur les plug-ins.

Le projet Eclipse Test & Performance Tools Platform (TPTP), anciennement appelé **Hyades**, est un ensemble de plug-ins Eclipse développés en licence EPL¹⁰. Il a été lancé en 2002, et est soutenu par des grands acteurs industriels tels que Intel, IBM, Computer Associates, Scapatech, etc.

La plate-forme TPTP offre quatre fonctionnalités principales.

- *Le monitoring* : TPTP fournit un ensemble d'agents pour collecter des informations systèmes pour les environnements Linux et Windows. D'autres agents applicatifs existent également pour superviser les serveurs MySQL, Apache, Jonas et JBoss. L'administration

⁸ Basic Encoding Rule

⁹ <http://www.eclipse.org>

¹⁰ Eclipse Public Licence

de ces agents se fait via un agent contrôleur « Remote Agent Controller » qui leur transmet les instructions envoyées par les clients (console Eclipse).

- *L'analyse des logs* : l'analyseur générique de TPTP (GLA = Generic Log Analyser) permet d'extraire et de convertir le contenu de n'importe quel fichier de logs vers un format pivot qui est le format CBE (Common Based Event). Nous revenons en détail sur ce module dans la deuxième partie de ce rapport.
- *Le profiling* : utilise l'interface JVMTI¹¹ fournie par la machine virtuelle Java pour collecter des mesures sur le nombre et la durée d'exécution d'une méthode ou d'une classe.
- *Le test* : permet de réaliser des tests unitaires et des tests de performances.

Le projet TPTP a l'avantage de bénéficier de la dynamique de la communauté Eclipse et de ses nombreux appuis dans le monde professionnel. Un des grands défis de ce projet est de concentrer différents types d'informations hétérogènes à différents niveaux (ressources systèmes, profiling, fichiers journaux, etc.) pour permettre de trouver une corrélation entre les données et simplifier le diagnostic.

2.2. Architecture de TPTP

TPTP adopte une architecture distribuée composée d'une partie cliente et d'une partie serveur, elle-même composée d'un agent contrôleur et d'un ensemble d'agents (Cf. Figure 15)

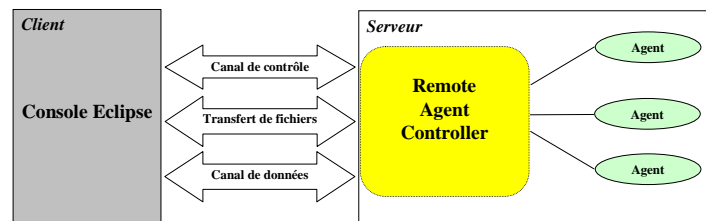


Figure 15 : Architecture de TPTP

Sur chaque machine supervisée est installé un agent contrôleur (RAC pour Remote Agent Controller) qui pilote les agents de supervision installés sur la même machine. Il existe trois types de communication entre la console Eclipse et le RAC qui utilisent trois canaux différents:

- *Canal de contrôle*: l'agent contrôleur écoute sur un port qui a été défini dans son fichier de configuration (10002 par défaut). Le client (console Eclipse) utilise ce port pour transmettre des commandes de supervision (démarrage, suspension, arrêt d'un agent). La

¹¹ Java Virtual Machine Test Interface

communication entre la console et l'agent contrôleur peut être sécurisée par le protocole SSL¹². Un autre port est alors défini pour cette option.

- *Canal de transfert des fichiers*: utilisé pour importer des fichiers depuis la machine cible (Ex fichiers journaux, etc.)
- *Canal de données*: le client transmet au RAC le port qu'il faut utiliser pour lui transmettre les informations collectées par les agents.

Le scénario typique des communications entre le client et le RAC est donné en ANNEXE C.

Le RAC est le composant principal de l'architecture de TPTP. En plus de la gestion des agents, le RAC peut analyser des fichiers hétérogènes locaux et transmettre leurs contenus sous forme de fragments CBE vers la console. Cette fonctionnalité d'analyse des fichiers existe également sous forme de module autonome appelé GLA (Generic Log Adapter). Ce module peut être exécuté en ligne de commande et dispose d'un fichier de configuration qui facilite son paramétrage.

2.3. Atouts et faiblesses de la plate-forme TPTP

Nous avons vu que l'architecture distribuée de TPTP est hiérarchisée. Cela veut dire que le client communique avec les agents par l'intermédiaire de l'agent contrôleur contrairement au standard SNMP où la communication est directe entre le client et les agents. De plus, une partie des traitements appliqués aux informations collectées par les agents, est réalisée sur le RAC c'est le cas du filtrage et l'encodage en XML. Ces traitements en amont facilite l'analyse au niveau de la console Eclipse mais engendre une intrusivité du fait que le RAC, comme les agents, sont tous déployés sur le système supervisée et donc consomment davantage de ressources système pénalisant ainsi le fonctionnement du système.

Contrairement au standard SNMP, l'envoi des informations collectées se fait en mode asynchrone ; le client reçoit ces informations sans avoir à les demander à chaque fois. Cette approche permet de réduire les échanges entre le client et le RAC et donc de réduire éventuellement le niveau d'intrusivité au niveau du réseau.

Une des faiblesses de TPTP est qu'il est difficile de présenter dans le même graphe les informations collectées par plusieurs agents ce qui limite la possibilité de corrélation des résultats. Toutefois, il existe un recours possible pour contourner cette limite et qui consiste à exporter les résultats dans des fichiers pour les exploiter par les outils usuels (ex. tableaux

¹² Secure Socket Layer

Excel). Pour les mêmes raisons, l'intégration des informations générées par des sondes externes est difficile bien que la fonctionnalité d'import existe mais elle concerne uniquement l'import des fichiers journaux (informations textuelles) et les fichiers de profiling.

Au niveau synchronisation, TPTP considère le moment de réception de l'information par le client comme étant le nouveau horodatage. La précision de ce mode de fonctionnement dépend fortement de la charge réseau (délais des transmissions) et des ressources systèmes disponibles pour la console Eclipse.

2.4. Discussion

La mise en place de tests et l'analyse des performances des applications sont des phases essentielles dans le cycle de développement. Le projet TPTP a pour but de répondre à ces besoins en proposant, d'une part, l'outillage nécessaire et en fournissant, d'autre part, un socle extensible pour analyser les fichiers générés par des outils externes à la plate-forme.

La supervision d'une nouvelle machine nécessite l'installation d'un agent contrôleur (RAC) et d'un ensemble d'agents. Le RAC pilote les agents et assurent d'autres tâches telles que le filtrage et la mise en forme des résultats. Ces différentes tâches requièrent des ressources systèmes ce qui augment l'intrusivité de la plate-forme TPTP.

Concernant l'intégration des informations générées par des outils externes, TPTP fournit la fonctionnalité d'import des fichiers mais qui est restreinte, malheureusement, à des fichiers journaux ou des fichiers de profiling et non pas à des fichiers de supervision comme ceux générés par la sonde *Vmstat* ou par l'outil *OpenSTA*.

TPTP fournit un module, fonctionnant en mode « Stand-alone », qui permet d'analyser et d'exporter des fichiers hétérogènes. Ce module appelé « GLA » pour « Generic Log Adapter » peut être un bon candidat pour fournir le socle de base du moteur de traitement qu'on souhaite intégrer dans notre framework de supervision. Le module « GLA » a l'avantage d'être générique et portable car il est développé en java. De plus, sa structure modulaire permet de le faire évoluer pour permettre des traitements personnalisés des informations.

3. NAGIOS

3.1. Présentation

NAGIOS est un outil de supervision open Source (Licence GPL) conçu pour fonctionner sous un système d'exploitation Linux (compatible UNIX). En plus de la fonction de collecte, de

stockage et de présentation, NAGIOS intègre un mécanisme de notification qui envoie des alertes si un des seuils prédéfinis a été dépassé (ex. Mémoire_{Consommée} > 80%).

Parmi les autres fonctionnalités qu'offre NAGIOS, on peut citer:

- Vérifier la disponibilité des équipements réseaux et des services (systèmes et applicatifs)
- Analyser les fichiers journaux
- Mesurer la consommation des ressources systèmes
- Effectuer des actions automatiques correctives

3.2. Architecture

L'architecture de NAGIOS est constituée d'une interface web, d'un moteur d'exécution (Core NAGIOS) et d'une bibliothèque de sondes (appelés plug-ins) déployées sur les machines supervisées (Cf. Figure 16).

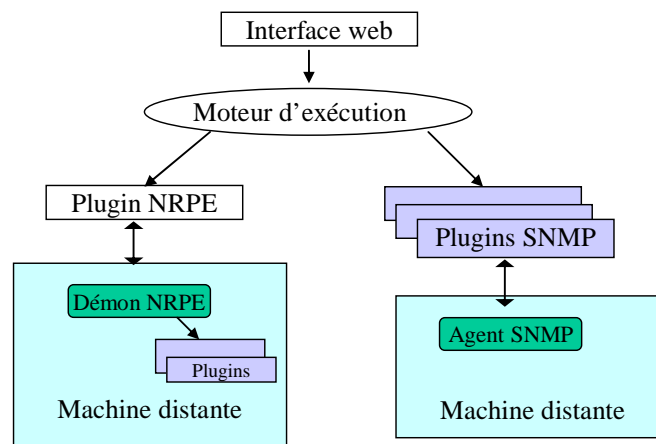


Figure 16 : Architecture de NAGIOS

Pour superviser une machine distante, on peut procéder de deux façons :

- Interroger un agent SNMP via des plug-ins SNMP personnalisés qui se trouvent sur la même machine que NAGIOS. Une variété de ces plug-ins est disponible sur le site de Sourceforge¹³

¹³ <http://sourceforge.net/projects/nagios-snmp/>

- Utiliser un démon NRPE (NAGIOS Remote Plug-in Executor) qu'il faut installer sur chaque machine supervisée. Ce démon joue le même rôle qu'un agent contrôleur (Cf. Architecture TPTP) qui exécute les plug-ins sous sa charge suite à une demande émanant du moteur d'exécution. Le plug-in NRPE (implémenté dans le fichier *check_nrpe*) permet de gérer les paramètres de communication avec le démon NRPE et de gérer les arguments de supervision (ex seuils d'alertes, etc.)

NAGIOS bénéficie d'une communauté très active qui enrichit régulièrement la bibliothèque des « plug-ins ». Aucune contrainte n'existe à priori pour développer un plug-in, ni sur le choix du langage ni sur les interfaces à implémenter. La seule condition est que le code retour du plug-in doit être un entier compris entre 0 et 3 (Cf. ANNEXE D)

3.3. Atouts et faiblesses de la plate-forme NAGIOS

NAGIOS communique avec ses plug-ins suivant deux modes :

- Mode passif: pour assurer une supervision événementielle (Event Based Monitoring). Cela permet de surveiller, par exemple, l'occurrence d'un événement (ex : violation d'une politique de sécurité). En terme d'intrusivité, ce mécanisme générerait moins de trafic sur le réseau, car seules les informations utiles sont transmises en plus du caractère unidirectionnel de la communication entre le noyau de NAGIOS et la machine supervisée. Cependant, la mise en place de ce mécanisme n'est pas aisée et nécessite de surveiller en permanence l'activité du système pour détecter les événements d'intérêts.
- Mode actif : pour assurer une supervision périodique (Time Based Monitoring). Il s'agit d'une communication synchrone (Cf. Figure 16) où le noyau de NAGIOS exécute, via le démon NRPE, les plug-ins distants et récupère le résultat. Côté intrusivité, le mode actif générerait plus de trafic que le mode passif vu que la collecte d'une information implique l'envoi d'une requête et la réception de la mesure.

Concernant l'intégration des mesures produites par des outils externes, NAGIOS ne propose pas de mécanisme pour le faire. De plus, la structure des messages manipulés par ce framework est peu flexible et ne permet pas de répondre à cette exigence.

Quant à l'exigence de découplage entre la collecte et la présentation, il existe des modules qui permettent de stocker les informations collectées par NAGIOS dans des bases cycliques de type RRD¹⁴ (outil NAGIOSGraph¹⁵) ou dans des bases relationnelles de type MySQL (outil

¹⁴ Round Robin Database

¹⁵ <http://nagiosgraph.sourceforge.net> solution frontale, pour nagios et développée en perl, qui permet de générer des graphiques

PerfParse¹⁶). Cependant, ces deux outils ne permettent pas d'insérer des traitements personnalisés entre le stockage et la présentation. Ceci dit, la conception des deux outils n'offre pas le découplage nécessaire pour traiter les informations avant de les présenter.

3.4. Discussion

NAGIOS est un framework de supervision doté de mécanismes de notifications pour envoyer des messages dès qu'un comportement anormal est détecté. Son architecture est hiérarchique et nécessite le déploiement d'un démon sur chaque machine supervisée. Ce démon a la tâche d'exécuter périodiquement les sondes locales et d'envoyer les informations collectées vers le moteur d'exécution de NAGIOS.

La communication entre le démon NRPE et le moteur d'exécution est synchrone ce qui augmente le trafic échangé sur le réseau. De plus, l'architecture « pseudo »-fermée de NAGIOS ne permet pas d'insérer des traitements supplémentaires entre la phase de collecte et la phase de présentation. De plus, il est difficile d'adapter NAGIOS pour qu'il puisse prendre en charge les informations collectées par des outils externes (i.e. résultats d'*OpenSTA*). Malgré les limites du framework NAGIOS, son étude a permis d'identifier diverses techniques/scripts pour collecter les informations sur l'utilisation des ressources système et sur la charge du réseau.

4. LeWYS (Lewys is Watching Your System)

4.1. Présentation

Nous clôturons cet état de l'art avec un projet issu de la recherche académique sur les modèles de composants. LeWYS est une bibliothèque de sondes développées à l'aide du modèle FRACTAL (Cf. ANNEXE E). L'objectif de ce projet est de fournir un canevas générique pour construire des systèmes de supervision de tailles variables.

LeWYS est implémenté en java en utilisant les API Julia et s'appuie sur la bibliothèque DREAM (Cf. ANNEXE E) pour diffuser les informations collectées par les sondes. Cette bibliothèque s'intègre plus facilement dans LeWYS car tous les deux sont basés sur le même modèle de composant FRACTAL.

LeWYS tire profit du modèle FRACTAL pour proposer un nouveau concept qui devrait faciliter la construction de plates-formes de supervision pour des systèmes de tailles quelconques. La

¹⁶ <http://perfparse.sourceforge.net/>

composition hiérarchique et la reconfiguration dynamique sont les deux caractéristiques principales mises en avant par le modèle FRACTAL et qui peuvent faciliter l'activité de la supervision :

- La composition hiérarchique devrait permettre de construire plus facilement des sondes composites (à grand grain) et des sondes génériques qui peuvent s'adapter aux différents contextes de supervision [Himdi.'04].

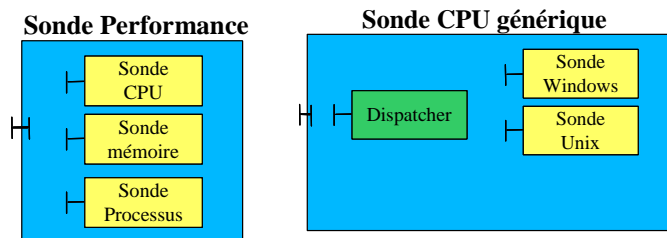


Figure 17 : Des exemples d'une sonde composite et d'une sonde générique

- La reconfiguration dynamique devrait permettre la dé/connexion à chaud d'une sonde ou un autre composant (ex filtre) afin de s'adapter au contexte de la supervision (Cf. Figure 18).

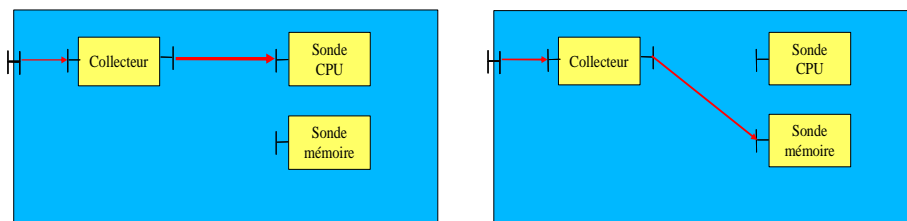


Figure 18 : Exemple illustrant la reconfiguration dynamique

4.2. Architecture

L'élément central dans l'architecture de LeWYS s'appelle une pompe « Pump », qui est un composant composite regroupant un ensemble de sondes et de contrôleurs. Le schéma de la Figure 19 donne une vue récursive de l'architecture d'une pompe LeWYS.

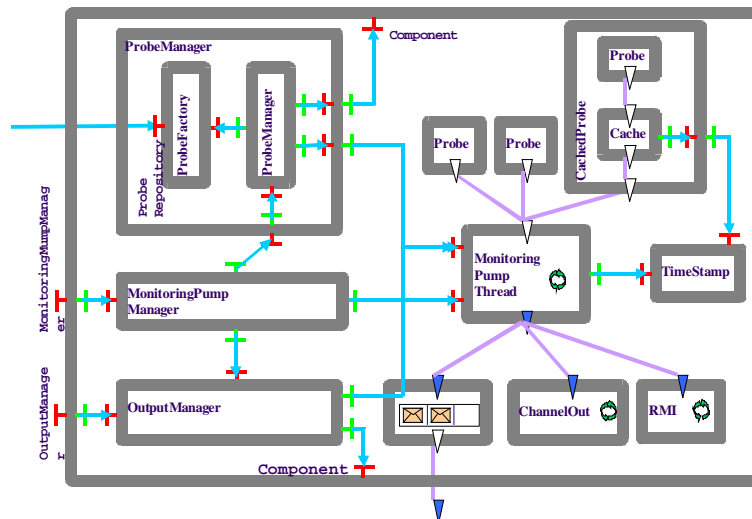


Figure 19 : Architecture d'une pompe

- **ProbeManager** : dans ce composant composite on distingue deux sous-composants: le *ProbeFactory* qui crée les instances des sondes et le *ProbeManager* qui les gère.
- **MonitoringPumpThread (multiplexeur)**: c'est le composant principal de la pompe. Il possède des interfaces clientes et serveurs :
 - ✓ Les interfaces clientes «pull» sont reliées à un ensemble de composants sondes. le multiplexeur récupère, par le biais de ces interfaces, les messages produits par une ou plusieurs sondes
 - ✓ Les interfaces serveurs «push» sont reliées à un ensemble de canaux de communication, ce qui permet au multiplexeur de transmettre les messages à un de ces canaux.
- **MonitoringPumpManager** : il permet aux utilisateurs d'interagir avec le multiplexeur; il implémente une interface qui permet de souscrire aux informations de supervision d'une ou plusieurs sondes et permet de spécifier un ensemble de canaux auxquels ces informations doivent être envoyées.
- **OutputManager** : ce gestionnaire permet d'insérer et de retirer dynamiquement des canaux de transmissions et de les lier au multiplexeur.
- **Probe/CachedProbe** : LeWYS définit deux types de sondes. Les sondes sans cache (*Probe*) qui envoient directement les données collectées vers le composant *MonitoringPumpThread*, puis les sondes qui gèrent un cache (*CachedProbe*)

Une pompe est déployée sur chaque machine observée, les informations collectées peuvent être ensuite transmises via des canaux de communication construits à base de composants DREAM.

4.3. Atouts et faiblesses de la bibliothèque LeWYS

La bibliothèque LeWYS est construite à base de composants FRACTAL ce qui lui apporte une certaine généralité dans son architecture. Les sondes LeWYS fournies actuellement sont constituées principalement de sondes pour les systèmes d'exploitation (Linux/Windows) et de sondes applicatives.

- Les sondes Linux récupèrent les informations de performance en se basant sur les systèmes de fichiers montés sous l'entrée `/proc`.
- Les sondes Windows accèdent aux informations relatives aux performances du système à travers la base de registre (sous la clé `HKEY_PERFORMANCE_DATA`)
- Les sondes applicatives sont des sondes JMX¹⁷ dont le but est de collecter des informations dans les environnements java (Cf. Figure 20).

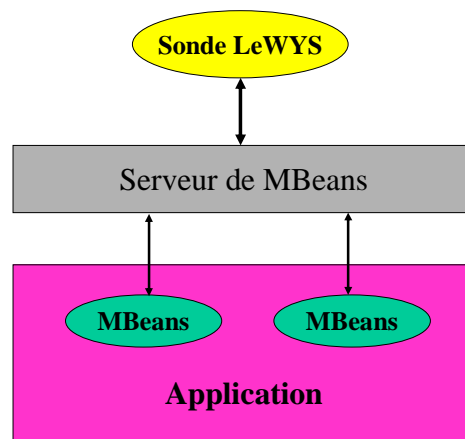


Figure 20 : Architecture JMX

Paradoxalement, une des grandes faiblesses de LeWYS est son couplage fort avec le projet DREAM ce qui rend difficile sa maintenabilité et donc la possibilité de pouvoir l'évoluer indépendamment de ce dernier.

4.4. Discussion

Actuellement, le projet LeWYS n'est plus actif pour deux principales raisons : une dépendance forte avec le projet DREAM et l'absence d'une politique claire d'évolution du projet. Néanmoins, nous soulignons l'intérêt de cette réflexion qui vise la construction d'une plate-

¹⁷ Java Management Extensions

forme de supervision à l'aide du modèle de composant FRACTAL. Ce dernier propose un modèle générique avec des APIs d'implémentation notamment celles en Java avec les APIs Julia¹⁸. Les deux caractéristiques principales qu'offre ce modèle sont d'une part la composition hiérarchique qui permettrait de construire des composants récursifs et d'autre part la reconfiguration dynamique qui permettrait de modifier, en cours d'exécution, la structure du framework de supervision afin de s'adapter à un nouveau contexte de supervision (modification de sondes, ajout du filtrage, etc.)

Concernant les trois exigences que nous étudions dans cette thèse, il est difficile d'y répondre convenablement avec le minimum fourni dans LeWYS. Néanmoins, le but de ce projet est d'illustrer l'applicabilité de FRACTAL dans le domaine de la supervision comme il l'est pour d'autres domaines, tout en mettant l'accent sur l'aspect architectural d'une plate-forme construite autour de LeWYS.

5. Synthèse

Le but de ce chapitre est de montrer le contexte technologique de la thèse, et en particulier les principaux standards et plates-formes de supervision. En plus du standard SNMP, nous avons étudié trois autres solutions qui nous semblent représentatives du reste et qui se distinguent par leurs architectures et leurs modes de fonctionnement. Ces solutions sont la plate-forme TPTP, le framework NAGIOS et la bibliothèque LeWYS.

A travers une étude comparative, nous exposons les principaux atouts et faiblesses des ces solutions et nous discutons la possibilité de répondre totalement ou partiellement aux trois exigences que nous avons fixé, à savoir : la faible intrusivité, la possibilité d'intégrer des sondes hétérogènes et le découplage entre l'acquisition et la présentation des informations collectées.

La plate-forme TPTP fournit une solution globale aux diverses fonctionnalités allant du domaine du test jusqu'à l'activité de monitoring et de profiling des applications java. La fonctionnalité de monitoring, qui nous intéresse en particulier, est assurée par un ensemble d'agents capables de mesurer l'activité des systèmes Linux et Windows. Ses agents sont déployés sur la machine observée et communiquent avec le client (console Eclipse) à travers le contrôleur d'agent appelé « RAC » qui est aussi installé sur la même machine. Ce dernier, assure plusieurs tâches dont le filtrage et la mise en forme des informations collectées par les

¹⁸ <http://fractal.objectweb.org/tutorials/julia/index.html>

agents. Ces différentes tâches requièrent des ressources systèmes ce qui augmente l'intrusivité de cette solution. Les deux autres exigences semblent difficilement réalisables et en particulier l'intégration des informations hétérogènes. En effet, le **monitoring** dans TPTP ne propose pas de mécanisme pour importer des fichiers et même si on développe un agent pour le faire, il serait difficile de gérer l'horodatage des informations contenues dans ces fichiers.

NAGIOS est un framework de supervision doté de mécanisme de notification. Il est utilisé, à la base, pour vérifier la disponibilité des services et des ressources systèmes. NAGIOS supporte deux modes de communications :

- Un mode passif où les plug-ins déployés sur la machine supervisée déclenchent une alerte suite à l'occurrence d'un événement. Ce mode, plutôt destiné à surveiller les fichiers journaux, a l'avantage d'utiliser moins de ressources réseaux mais il a l'inconvénient d'être difficile à mettre en place.
- Un mode actif où le noyau NAGIOS sollicite périodiquement ses plug-ins pour effectuer des mesures locales. Ce mode est plus facile à mettre en place mais utilise plus de ressources réseaux que le premier car la communication est bidirectionnelle entre le noyau NAGIOS et ses plug-ins.

Les messages échangés dans NAGIOS ont une structure peu flexible ce qui rend difficile l'intégration des informations collectées par des outils externes. Le « workflow » des informations est statique et ne permet pas d'insérer des traitements entre la phase de collecte et celle de présentation sauf si on applique un enregistrement intermédiaire des informations (dans une base de donnée par exemple) avant la phase d'exploitation.

En dernier lieu, nous avons étudié LeWYS, une bibliothèque de sondes construites à l'aide du modèle de composant FRACTAL. Ce modèle offre la possibilité de construire des architectures génériques et hiérarchiques supportant la reconfiguration dynamique. Ces deux caractéristiques peuvent être utiles pour la supervision et notamment la reconfiguration dynamique qui devrait permettre d'insérer des composants dynamiquement dans une architecture comme par exemple un composant de filtrage pour réduire la quantité des informations générées. Théoriquement, il est possible de répondre aux trois exigences qu'on s'est fixé en utilisant le modèle FRACTAL et en s'inspirant du projet LeWYS. Cependant, le manque de maturité de ce nouveau modèle et l'instabilité de LeWYS sont des raisons parmi d'autres qui ont poussés à ne pas retenir cette solution.

Le Tableau 2 présente une synthèse du degré d'adéquation entre les solutions présentées précédemment et les trois objectifs que nous avons fixé:

<i>Exigences</i>	SNMP	TPTP	NAGIOS	LeWYS
<i>Intrusivité</i>	Utilisation des ressources réseaux	Consommation des ressources systèmes par les agents	Peu intrusive	Peu intrusive
<i>Intégration des sondes hétérogène</i>	Non	existe mais pas pour le monitoring	Difficile à mettre en œuvre	Difficilement réalisable
<i>Découplage entre la collecte et la présentation</i>	Non	Couplage fort entre la collecte et la présentation	Workflow statique → découplage difficile	Difficilement réalisable
<i>Constat général</i>	Solution générique mais incomplète	Projet actif mais n'offre pas le découplage entre collecte et présentation	Projet actif mais difficile à adapter à nos objectifs	Solution incomplète et projet inactif

Tableau 2 : Comparatif des solutions de supervision vis à vis des exigences

Ce tableau démontre les limites des quatre solutions de supervision à satisfaire les exigences qu'on s'est fixées. En effet, l'intégration des sondes nécessite l'existence d'un module générique capable d'extraire et d'analyser les informations contenues dans des fichiers hétérogènes. La plate-forme TPTP fournit le module GLA qui peut être adapté pour remplir cette fonction. Ce module est développé en java et existe sous forme d'un composant autonome qui peut être intégré facilement dans n'importe quelle plate-forme développée en java.

Dans le chapitre suivant, nous décrivons la conception de la plate-forme et nous exposons notre démarche pour répondre à ces trois exigences avec un seul framework de supervision.

PARTIE 3
CONCEPTION & MISE
EN OEUVRE

D. Conception

1. Rappel de la problématique

Dans la première partie de ce manuscrit, nous avons établi une liste d'exigences qui nous semblent particulièrement utiles pour superviser des applications distribuées hétérogènes. Pour rappel, ces exigences sont les suivantes :

- La faible intrusivité: car c'est un gage de la fiabilité des mesures collectées. L'activité de supervision ne doit pas (ou très peu) perturber le fonctionnement de l'application supervisée.
- L'intégration des sondes hétérogènes : il est souvent nécessaire de combiner plusieurs outils pour évaluer globalement les performances d'une application (temps de réponse, utilisation des ressources, etc.). Notre outil de supervision doit permettre de gérer la diversité et l'hétérogénéité des informations collectées par ces outils.
- Le découplage entre la phase de collecte et la phase de présentation ce qui permet d'appliquer des traitements sur les informations collectées avant de les afficher.

Nous avons étudié, chacune de ces exigences, à la lumière des travaux de recherche et des solutions technologiques. Il résulte de cette étude quelques consignes pouvant aider à la conception d'un framework de supervision répondant aux exigences citées précédemment :

Dans ce chapitre, nous exposons notre démarche pour répondre aux exigences et nous allons décrire l'architecture de notre plate-forme ainsi que la conception des différents modules qui la constituent.

2. Description du framework de supervision

2.1. Introduction

Rappelons d'une manière générale les fonctionnalités principales d'un framework de supervision :

- La collecte des informations : nécessite le déploiement d'un ou plusieurs composants (sondes ou agents) sur les machines supervisées.
- Le transfert (ou diffusion) : peut prendre plusieurs formes (Sockets, MOM, FTP, etc.).

- Le traitement : regroupe toutes les opérations appliquées aux informations avant leur présentation.
- La présentation des informations sous différents formats (graphes, tableaux, html, etc.).

En se référant aux consignes élaborées dans le paragraphe précédent et en les projetant sur les 4 fonctionnalités citées précédemment, nous proposons les scénarios de fonctionnement suivants :

- Les sondes déployées sur les machines supervisées collectent les informations sur l'activité du système et stockent ces informations, non traitées, dans des fichiers locaux (fichiers des résultats). Aucune information n'est transmise sur le réseau ce qui réduit voire élimine le risque d'intrusivité au niveau des ressources réseaux.
- Les fichiers des résultats ainsi que d'autres fichiers produits par des sondes externes, éventuellement hétérogènes, sont ensuite analysés et transformés, par un moteur de traitement, dans un format homogène.
- La présentation des résultats est réalisée à partir des fichiers homogènes,
- Le découplage souhaité entre la collecte et la présentation est garanti par le stockage des informations dans des fichiers.

Dans la suite de ce manuscrit, nous décrivons d'une manière plus élaborée et à travers des diagrammes UML, l'architecture physique et logicielle ainsi que son fonctionnement.

2.2. Architecture physique

L'architecture physique de la plate-forme est décrite dans la Figure 21.

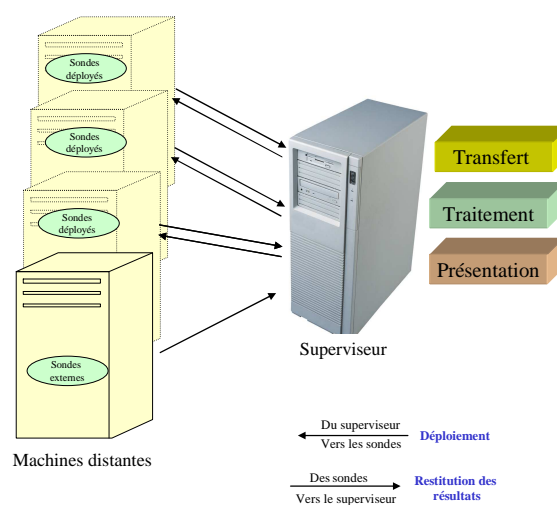


Figure 21 : Architecture physique de la plate-forme de supervision

L'architecture physique est composée d'une machine superviseur où sont déployés les modules suivants :

- Un module de transfert qui déploie les sondes sur les machines supervisées et qui récupère les fichiers des résultats pour les traiter localement.
- Un module générique pour un post traitement des fichiers des résultats (générés par les sondes ou par des outils externes).
- Un module de présentation graphique

2.3. Déroulement d'une supervision

Les Figure 22 et Figure 23 présentent respectivement les diagrammes d'activités et de séquences de la plate-forme de supervision.

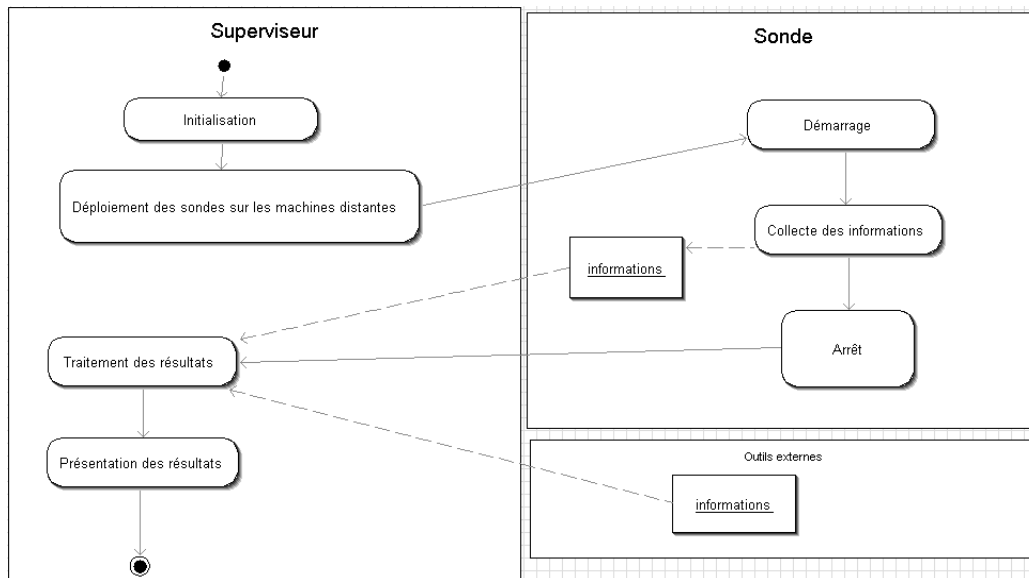


Figure 22 : Diagramme d'activités de la plate-forme de supervision

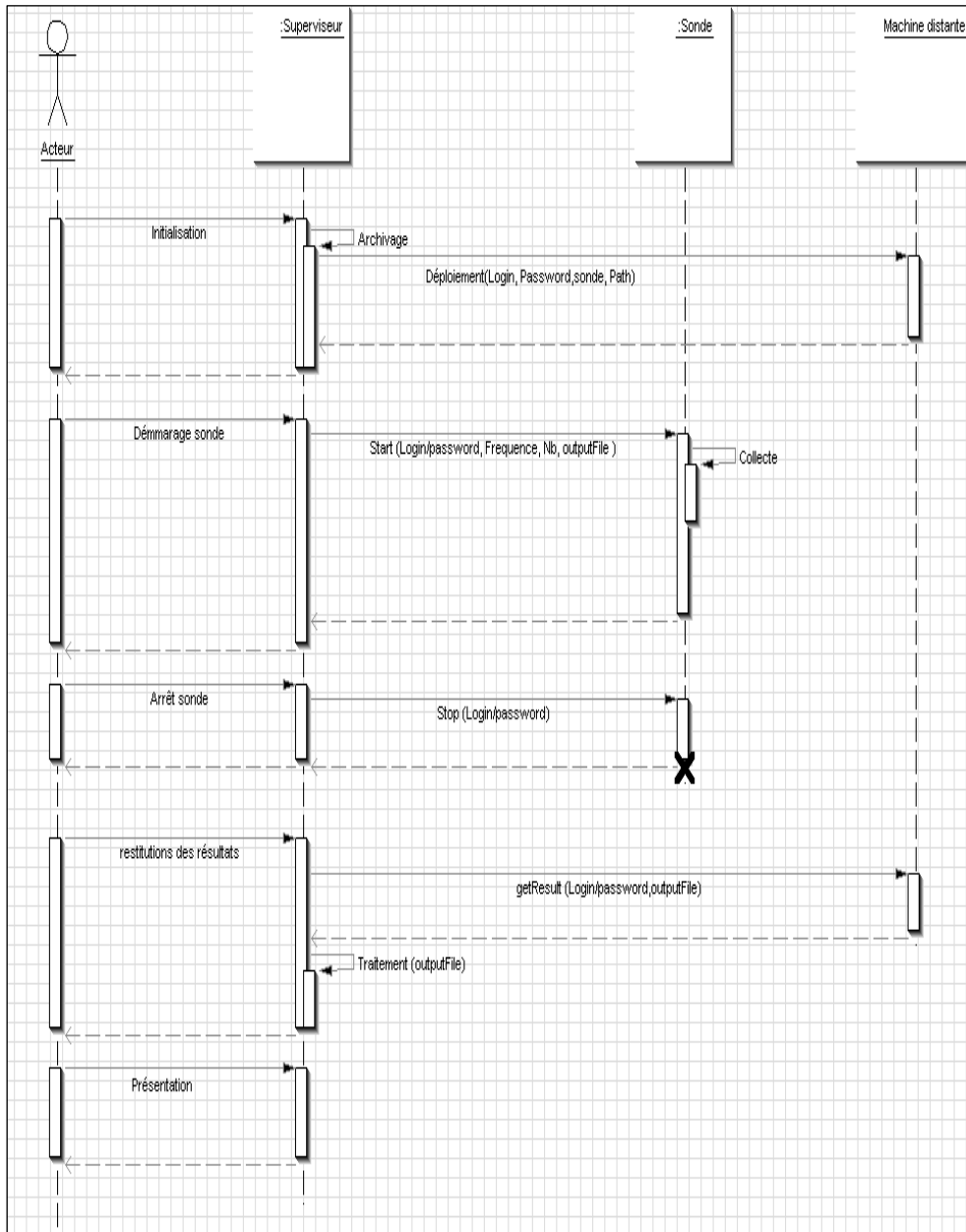


Figure 23 : Déroulement d'une supervision

Initialisation

La phase d'initialisation consiste à archiver les résultats (informations collectées, graphes, etc.) obtenus lors des supervisions antérieures. Durant cette phase, les instances d'adaptateurs (Cf. Paragraphe 3.4) décrivant les traitements appliqués aux fichiers de résultats sont créées.

Déploiement des sondes

Les sondes sont déployées sur les machines distantes dans des répertoires passés en argument. Ce déploiement nécessite d'abord de s'authentifier auprès de ces machines. Deux possibilités sont envisagées :

- Une authentification avec (*identifiant/mot de passe*)
- Une authentification forte avec des certificats.

Par souci de simplicité : la première solution est utilisée en supposant l'existence d'un compte commun pour accéder à toutes les machines supervisées.

Démarrage/arrêt des sondes

Le superviseur se connecte sur chaque machine en s'authentifiant avec le login /password et exécute chacune des sondes déployées en leur indiquant les arguments qu'elles requièrent :

- Le mode (*START* ou *STOP*),
- Intervalle de temps, en seconde, entre deux mesures. Cet intervalle représente pour nous la fréquence de collecte,
- Le nombre de mesures réalisées (Nb). On peut alors déduire la durée de la supervision en multipliant ce nombre par la fréquence de la collecte,
- Le chemin absolu vers le fichier de sortie qui va contenir les mesures collectées (paramètre *outputFile* dans la Figure 23)

Tous ces arguments sont paramétrables et sont déclarés dans un fichier de configuration.

La collecte d'information est terminée si :

- Le nombre d'échantillons, passé en argument à la sonde, a été atteint
- La sonde est ré-exécutée avec l'option « stop ».

Restitution & traitement des résultats

Cette étape comprend plusieurs phases :

- *Mise en forme des résultats*

En fin de collecte, le fichier de résultats produit par la sonde, est récupéré vers le superviseur en vue de son traitement. On utilise le même mécanisme de transfert des fichiers pour le déploiement des sondes et pour le transfert des résultats. Le module de traitement (Cf. page 74) utilise l'instance d'adaptateur, associée à la sonde et créée dans la phase d'initialisation, pour exporter le fichier de résultat, généré par la sonde, dans un format structuré qui est le CBE. Le même processus de mise en forme est appliqué aux fichiers générés par des outils tiers, initialement hétérogènes, ce qui permet d'homogénéiser leurs formats (Cf. Figure 24).

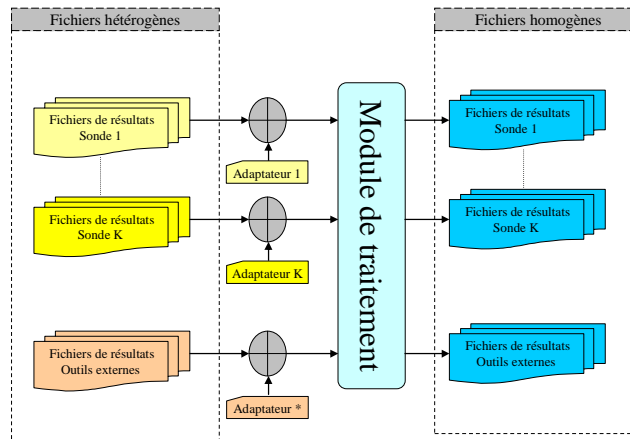


Figure 24 : Mise en forme des fichiers de résultats

○ *Synchronisation des résultats*

Comme nous l'avons souligné dans la première partie de ce rapport, notre plate-forme de supervision utilise un mécanisme de synchronisation en aval basée sur une translation de l'horodatage des fichiers. Nous utilisons l'horloge de la machine superviseur comme horloge de référence puis nous mesurons les décalages d'horloges entre les machines supervisées et cette machine de référence. L'horodatage des résultats de supervision est recalculé en tenant compte de ces décalages qui sont représentés par le symbole ($\Delta_{T_0-T_K}$) dans la Figure 25 avec T_0 l'heure de référence.

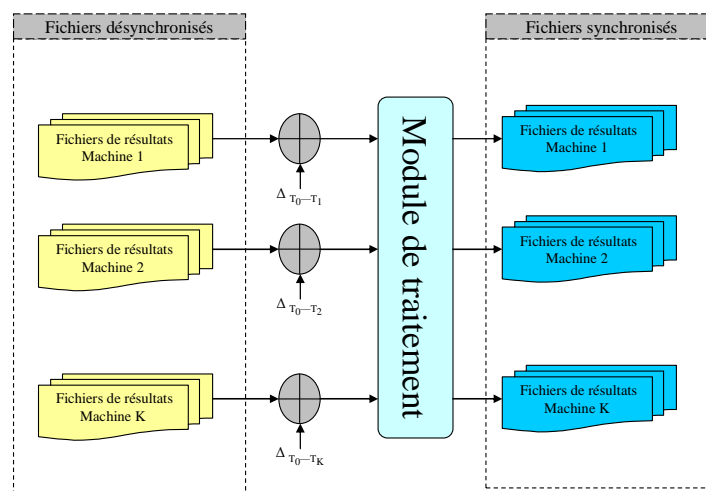


Figure 25 : Synchronisation par décalage temporel

($\Delta_{T_0-T_K}$) correspond au décalage entre l'horloge de la machine K et l'horloge de référence. Ce décalage peut être évalué visuellement en comparant les deux horloges (l'horloge distante et

l'horloge de référence). Cette méthode ne permet pas d'obtenir une très grande précision sur le décalage. Néanmoins, elle peut être une bonne approche si on ne cherche pas une synchronisation quasi-parfaite.

Nous avons exploré une méthode basée sur l'interrogation à distance d'un serveur de temps installé sur les machines supervisées. Cette technique offre une meilleure précision des décalages temporels contrairement à la première technique jugée approximative.

Présentation des résultats

Au terme de la phase de traitement, la phase de présentation a pour objectif de permettre une exploitation facile des informations collectées. Généralement, les représentations graphiques sont les mieux adaptées pour avoir une vue de l'ensemble des résultats et faciliter la tâche de corrélation et de diagnostic.

Le module de présentation prend en entrée des fichiers homogènes qu'il analyse pour extraire les valeurs des métriques à représenter. Le format adopté pour les fichiers en sortie est le CSV (Comma-separated values) dans lequel les valeurs sont séparées par des virgules.

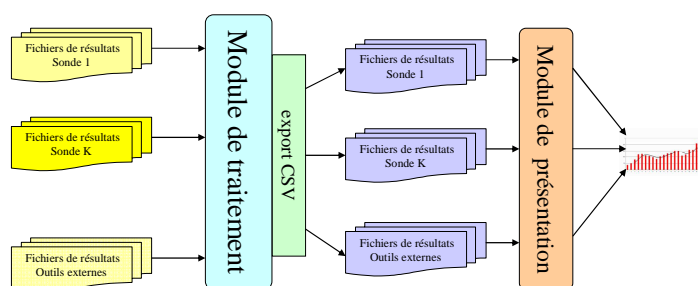


Figure 26 : processus de génération des graphiques

Il est possible de corréler les courbes de plusieurs métriques (ex. Cpu, mémoire, débit in/out, etc.) pour mieux analyser le comportement du système supervisé. La (Figure 27) illustre l'exemple d'une corrélation entre l'activité du processeur (consommation CPU) sur une machine et le nombre d'utilisateurs accédant à une application hébergée sur cette même machine.

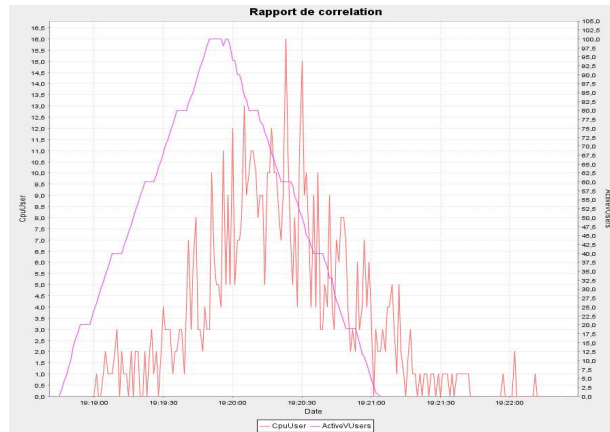


Figure 27 : Exemple de corrélation entre plusieurs paramètres

2.4. Architecture logicielle de la plate-forme

Les digrammes de déploiement montrent la localisation physique des matériels qui composent l'environnement de supervision et la répartition des composants de la plate-forme de supervision sur ces matériels.

La Figure 28 représente le diagramme de déploiement correspondant à l'état initial de la plate-forme de supervision.

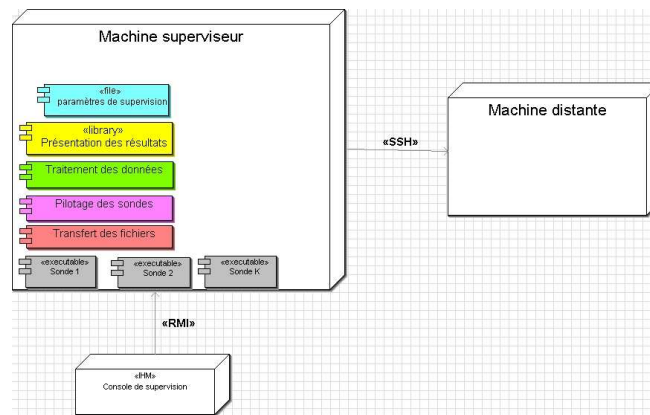


Figure 28 : Diagramme de déploiement (avant supervision)

La console de supervision est optionnelle et sert uniquement à exécuter à distance les tâches de supervision en utilisant le protocole RMI (Remote Method Invocation). Ces tâches de supervision sont implémentées sous forme de tâches Ant ce qui permet de les exécuter manuellement ou via des outils tiers.

Les communications entre la machine superviseur et les machines distantes (supervisées) sont établies via le protocole SSH¹ et nécessitent une phase d'authentification.

En fin de supervision, la nouvelle organisation des modules et des fichiers de la plate-forme est celle présentée dans la Figure 29

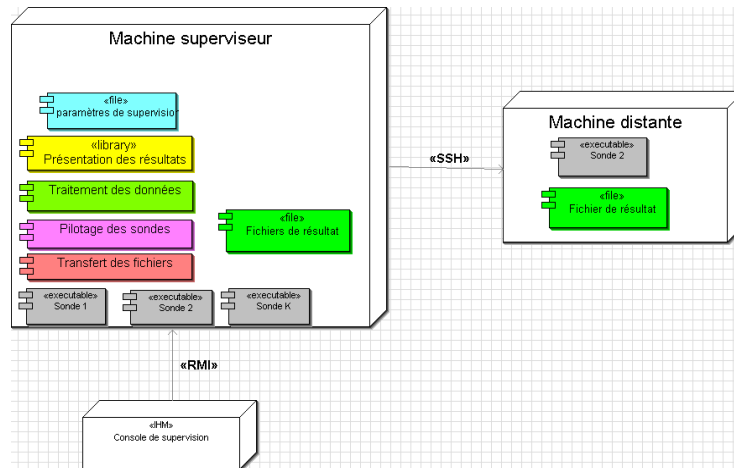


Figure 29 : Diagramme de déploiement (après supervision)

Les fichiers de résultat sont récupérés dans un répertoire localisé sur la machine superviseur. Ce répertoire, utilisé pour stocker d'autres fichiers de la plate-forme, se décompose en trois sous répertoires :

- *Input* : pour contenir les fichiers de résultats produits directement par les sondes
- *Output* : pour contenir les fichiers de résultats après traitement.
- *Config* : pour contenir les fichiers adaptateurs nécessaires pour le traitement des résultats

2.5. Description des composants de la plate-forme

Dans ce paragraphe nous décrivons les principales composantes de la plate-forme à savoir la structure d'une sonde de performance et la conception du moteur de traitement.

2.5.1. Les sondes

La plate-forme de supervision utilise un ensemble de sondes pour collecter les informations relatives aux performances du système. Certains frameworks de supervision utilisent au

¹ Secure SHell

contraire des agents pour collecter ces informations. Ces deux paradigmes (sonde/agent) peuvent parfois créer confusion vu qu'ils sont utilisés sans restriction : une sonde (*probe* en anglais) est un programme utilisé pour investiguer l'état d'un système ou d'une application, alors qu'un agent est un processus qui s'exécute en tâche de fond et qui déclenche une action suite à un événement ou à une sollicitation.

De part son fonctionnement, une sonde peut être moins intrusive qu'un agent car elle ne s'exécute que s'il y a besoin de mesurer une métrique. Un système de supervision basé sur les agents est adapté pour une supervision de type « *online* » contrairement à notre approche « *offline* » où les informations collectées par les sondes sont stockées localement dans des fichiers et restituées au terme de la phase de collecte pour être analysées. Avec cette approche, on espère réduire le trafic généré par l'activité de supervision et réduire ainsi l'intrusivité de la plate-forme de supervision.

Dans le paragraphe suivant, nous décrivons les caractéristiques d'une sonde de la plate-forme.

a. Conception d'une sonde

La plate-forme de supervision est conçue de manière à assurer un découplage entre les différents modules qui la constituent. Cette approche favorise la réutilisation de sondes de supervision existantes moyennant quelques adaptations. En effet, pour qu'elle soit intégrée et gérée par notre plate-forme de supervision, une sonde doit remplir les conditions suivantes :

- L'existence de deux modes : le mode (*Start*) pour démarrer la sonde et déclencher la collecte et le mode (*Stop*) pour arrêter la collecte.
- L'acceptation des arguments requis suivants : la fréquence de la collecte (Freq), le nombre d'échantillons à collecter (Nb) et le fichier où seront stockées les informations collectées (fileOutput).

En résumé, la structure typique d'une sonde de la plate-forme est représentée dans la Figure 30

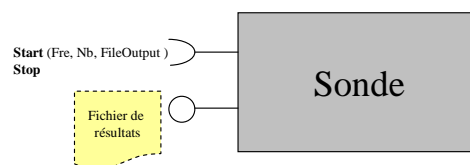


Figure 30 : Structure d'une sonde de la plate-forme

2.5.2. Moteur de traitement

Le cœur de la plate-forme est constitué d'un module qui a la charge d'analyser les fichiers de résultats, et de les transformer dans un format pivot commun. Ce module est basé sur le

composant appelé « GLA » pour « Generic Log Adapter » qui est un moteur générique d'analyse des fichiers journaux intégré dans la plate-forme TPTP.

a. Choix du « GLA »

Le choix du « GLA » comme moteur d'analyse est justifié par plusieurs raisons :

- Développement en Java (portabilité)
- Modularité
- Facilité d'extension
- Existence d'un éditeur graphique pour faciliter l'écriture des fichiers adaptateurs².

b. Structure du « GLA »

La décomposition du moteur d'analyse en sous modules est présentée dans la Figure 31

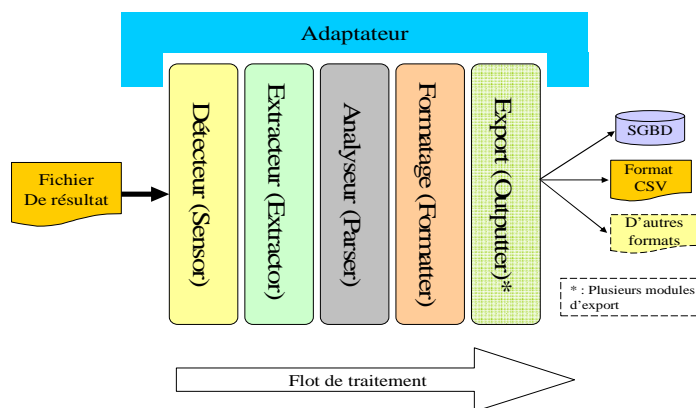


Figure 31 : Structure interne du « GLA »

- Le détecteur fournit le mécanisme de lecture du contenu du fichier de résultats en vue de son traitement.
- L'extracteur extrait plusieurs lignes d'entrées fournies par le détecteur et les sépare en frontière de messages.
- L'analyseur prend les messages délimités par l'extracteur et génère un ensemble de mappages de valeurs vers la structure de données CBE (Cf. ANNEXE F).

² Un adaptateur est un fichier, utilisé par le « GLA », qui décrit les traitements à effectuer pour mettre dans un format uniforme le contenu d'un fichier de résultat

- Le module de formatage effectue le mappage des attributs vers leurs valeurs fournies par le programme d'analyse syntaxique et compile l'instance d'objet Java appropriée
- Le module de sortie (*Outputter*) exporte les enregistrements CBE résultants, fournis par le module de formatage. Il est possible de définir plusieurs modules de sorties et de disposer ainsi des résultats de supervision sous différents formats. Nous avons choisi d'exporter les fichiers de résultats dans le format CSV (Comma Separated Value) et nous avons développé le module de sortie adapté à cet effet.
- *L'adaptateur* est le fichier où on déclare tous les paramètres liés au traitement des fichiers (ex nom du fichier à traiter, valeur du décalage temporel, etc.) et les noms des classes java associées à chaque sous-module.

La modularité de ce moteur de traitement permet de faire évoluer, d'une manière indépendante, chaque sous-module. Nous avons étendu les fonctionnalités du sous-module d'analyse « *parseur* » pour permettre la translation des échelles temporelles des fichiers de résultats. Grâce à cette fonctionnalité, il est plus facile de corréler des résultats générés sur des machines désynchronisées.

3. Conclusion

Dans ce chapitre nous avons décrit l'architecture et le fonctionnement de la plate-forme de supervision. Sa conception vise à répondre aux exigences prédéfinies :

- L'intrusivité au niveau des ressources réseaux est réduite voire éliminée en enregistrant les informations collectées dans des fichiers locaux sur la machine supervisée. Par conséquent, aucune information n'est transmise sur le réseau durant la phase de collecte. Au terme de cette phase, ces fichiers sont récupérés sur une machine dédiée pour les traiter.
- Les informations collectées sont enregistrées dans leurs états bruts, sans aucun traitement, pour réduire la consommation des ressources systèmes.
- La plate-forme dispose d'un moteur générique de traitement qui prend en entrée des fichiers sous des formats hétérogènes et les transforme dans un format commun structuré qui est le CBE. Ce moteur, qui est une adaptation de l'outil « GLA : Generic Log Adapter » de la plate-forme TPTP, dispose d'un ensemble de fonctionnalités dont la fonctionnalité d'export qui permet d'avoir les fichiers des résultats dans des formats faciles à exploiter (ex. Format CSV).
- La supervision se découpe en quatre tâches principales :
 - La collecte assurée par des sondes choisies pour leur faible degré d'intrusivité.

- Le transfert des fichiers de résultats
- L'analyse, la mise en forme et l'export de ces fichiers par le moteur de traitement
- La présentation des résultats

Le découplage entre les phases de collecte et de présentation est mis en œuvre au moyen du stockage des résultats dans des fichiers. La tâche de présentation prend en entrée des fichiers contenant des valeurs horodatées et non pas des données transmises directement par les sondes. Ce découplage permet à la fois de prendre en charge des résultats provenant d'autres outils de mesures et aussi d'appliquer des traitements sur ces résultats comme par exemple la synchronisation en aval des informations par décalage de l'échelle temporelle.

Nous avons décrit la structure d'une sonde de supervision et les arguments qu'elle requière pour son fonctionnement à savoir l'intervalle entre deux mesures et le fichier qui va contenir les informations collectées. Les sondes fonctionnent en mode autonome et aucune interaction avec la machine superviseur n'est prévue au cours de la phase de collecte ce qui permet de réduire les échanges entre ces deux éléments (sonde et superviseur).

L'autre composant détaillé dans ce chapitre est le moteur de traitement qui, comme nous l'avons déjà souligné, est basé sur le module « GLA » de TPTP. Ce module possède une architecture modulaire et dispose de nombreuses fonctionnalités dont la fonction d'analyse « Parsing » et d'export « Outputter »

Le chapitre suivant décrit la mise en œuvre d'un prototype de sonde de supervision. Ce prototype utilise la commande *Vmstat* et permet de mesurer l'utilisation des ressources systèmes dans un environnement Unix. Cet exemple devrait servir de modèle pour développer d'autres sondes comme par exemple une sonde basée sur la commande *ifstat*³ pour mesurer l'activité du réseau.

³ <http://www.linux-kheops.com/doc/man/manfr/man-html-0.9/man8/netstat.8.html>

E. Mise en oeuvre

1. Introduction

Ce chapitre est dédié à la description de l'implémentation de la plate-forme de supervision. Cette dernière peut être connectée à des outils (ex. gestionnaires de test) qui intègre l'activité de supervision.

Dans ce chapitre nous traitons deux points :

- *La description des sondes utilisées dans la plate-forme. Ces sondes sont « Platform-specific » et donc dépendent des systèmes sur lesquels elles sont déployées.*
- *La description du cœur de la plate-forme qui assure le reste des tâches de supervision (ex déploiement des sondes, traitement, présentation, etc.).*

Les sondes peuvent être développées dans n'importe quel langage à condition qu'il soit possible de les exécuter à partir de programmes java. Pour notre cas, la sonde *Vmstat* est développée en Script Shell. Le reste de la plate-forme est développé en java et est organisé sous forme de tâches Ant. Ce choix assure une portabilité et une flexibilité de la solution. D'autant plus que cette organisation sous forme de tâches Ant facilite l'automatisation de la supervision et l'intégration dans d'autres plates-formes comme des gestionnaires de test.

2. Implémentation d'une sonde

Dans ce paragraphe, nous illustrons l'implémentation possible d'une sonde système qui permet de mesurer l'activité d'un système Unix et qui utilise la commande *Vmstat*. Ce choix de sonde n'est pas arbitraire. En effet, *Vmstat* est une référence dans le domaine de supervision pour observer l'activité d'un système Linux. De plus, il existe d'autres commandes qui s'exécutent de la même façon que la commande *Vmstat* ce qui facilite l'adaptation de notre exemple à ces commandes (ex : *Ifstat*, *Iostat*, etc.)

2.1. Sonde Vmstat

2.1.1. Description de la sonde

Le programme *Vmstat* fournit des informations à propos des processus, de la mémoire, de la pagination, des interruptions et de la répartition du temps CPU.

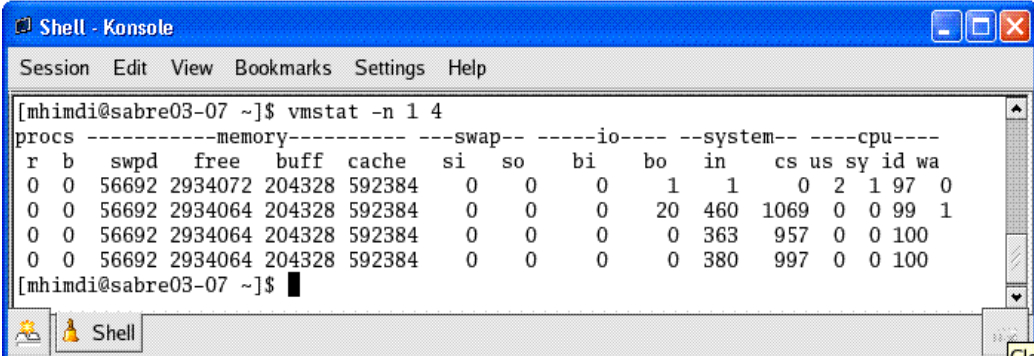
La commande Vmstat s'exécute avec les options suivantes :

```
vmstat [-n] [nb_sec] [total]
```

Figure 32 : Syntaxe de la commande Vmstat

- √ **-n** affiche, une seule fois, un en-tête décrivant les noms des métriques mesurées.
- √ **nb_sec** est l'intervalle d'échantillonnage en secondes. Si aucune valeur n'est spécifiée, une seule mesure sera affichée.
- √ **total** est le nombre de mesures effectuées. Si elle n'est pas indiquée et si **nb_sec** est spécifié alors la collecte est infinie (sans interruption).

La copie d'écran ci-dessous illustre l'utilisation de la commande *Vmstat* réalisée sur une machine Linux Fedora (FC4)



```
Shell - Konsole
Session Edit View Bookmarks Settings Help
[mhimdi@sabre03-07 ~]$ vmstat -n 1 4
procs -----memory----- --swap-- ----io----- --system-- -----cpu-----
 r b swpd free buff cache si so bi bo in cs us sy id wa
 0 0 56692 2934072 204328 592384 0 0 0 1 1 0 2 1 97 0
 0 0 56692 2934064 204328 592384 0 0 0 20 460 1069 0 0 99 1
 0 0 56692 2934064 204328 592384 0 0 0 0 363 957 0 0 100
 0 0 56692 2934064 204328 592384 0 0 0 0 380 997 0 0 100
[mhimdi@sabre03-07 ~]$
```

Figure 33 : Exemple des informations collectées avec Vmstat

Dans cet exemple, les mesures sont effectuées toutes les secondes pendant une durée de quatre secondes. Le détail des mesures fournies par Vmstat est donnée en détail dans [ANNEXE G]et concerne principalement les aspects suivants :

- la consommation de la mémoire : mémoire vive, mémoire bufférisée, etc.
- l'utilisation du disque : accès en lecture/écriture.
- l'utilisation du processeur : en mode noyau, en mode utilisateur, en mode système et en mode « Idle ».

Suivant les systèmes Unix, les métriques mesurées par la commande *Vmstat* peuvent différer légèrement. L'exécution de la commande avec l'option « -h » permet de connaître les options d'utilisation.

2.1.2. Implémentation

Une première implémentation de la sonde a été réalisée en langage Shell pour une machine Linux de type *Debian*. Cette implémentation est détaillée dans la Figure 34.

```
#!/bin/sh

# -----
# Script pour la récupération des infos de l'outil vmstat dans un fichier
# pendant la durée spécifiée
#
# Syntaxe :
#   Démarrage : log_vmstat start interval count
#               interval = intervalle en seconde entre deux mesures
#               count = nombre d'itérations (la durée du monitoring
#                   correspond a interval x count)
#   Arrêt : log_vmstat stop
#
# Auteurs : Thierry Boulet - KEREVAL - thierry.boulet@kereval.com
#           Marouane HIMDI - KEREVAL - marouane.himdi@kereval.com
# Version : 1.1 - 01/03/2006
# -----
# S'adapter au système
BINGREP=`which grep`
BINPS=`which ps`
BINVMSTAT=`which vmstat`
BINPGREP=`which pgrep`

# =====

case "$1" in
'start')
    if [ $# -eq 4 ]; then
        if [ -s vmstat.pid ]; then
            PID=`cat vmstat.pid`
            ISRUNNING=`$BINPS -p $PID | $BINGREP $PID`
            if [ "${ISRUNNING}" ]; then
                echo "'log_vmstat' est en cours d'exécution. Utiliser 'log_vmstat
                stop' avant de démarrer un nouveau monitoring."
                exit
            fi
        fi

        mem=`awk '/MemTotal:/ { print $2 }' < /proc/meminfo`
        echo `date '+%m/%d/%Y %H:%M:%S'`, $2, $mem > $4
        echo -PROCS_R PROCS_B MEM_SWPD MEM_FREE MEM_BUF MEM_CACH SWAP_SI SWAP_SO IO_BI
        IO_BO SYS_INT SYS_CS CPU_USR CPU_SYS CPU_IDL CPU_WAIT >> $4
        nohup $BINVMSTAT $2 $3 | $BINGREP -v -E 'procs ---|swpd' >> $4 &
        echo "Vmstat started."
        $BINPGREP -u $USER -n vmstat > vmstat.pid

    else
        echo "Usage: $0 { start interval count outputFileName | stop }"
        exit
    fi
;;

# =====
'stop')
    if [ -s vmstat.pid ]; then
        PID=`cat vmstat.pid`
        kill -9 $PID 2> /dev/null
        rm -f vmstat.pid
    fi
;;

*)
    echo "Usage: $0 { start interval count outputFile | stop }"
;;

esac
```

Figure 34 : Implémentation de la sonde Vmstat

Dans la première partie de l'implémentation (bloc 1), on déclare les commandes Unix utilisées pour assurer le fonctionnement de la sonde. La deuxième partie concerne les actions entreprises durant le démarrage de la sonde : on vérifie d'abord l'absence d'une autre instance de la sonde

et ceci en vérifiant l'absence de processus à l'état « running » de *Vmstat*. Ensuite, la sonde inscrit dans la première ligne du fichier de résultat, les informations suivantes :

- la date de démarrage (début de collecte) : cette information est indispensable puisqu'elle permet d'horodater les mesures qui ne le sont pas à la base.
- l'intervalle de collecte
- la mémoire vive totale, ce qui permet de mesurer le ratio de la consommation de la mémoire.

Dans la deuxième ligne, la sonde stocke les noms des champs mesurés par la commande *Vmstat*. Les lignes restantes correspondent aux mesures collectées.

L'arrêt de la sonde s'effectue avec l'unique argument « Stop ». Cette opération supprime le processus créé par l'exécution de la commande *Vmstat*.

Finalement, le fichier de résultats générés par la sonde *Vmstat* est celui de la Figure 35.

```

04/02/2007 16:31:59,1,4137228
-PROCS_R PROCS_B MEM_SWPD MEM_FREE MEM_BUF MEM_CACH SWAP_SI SWAP_SO IO_BI IO_BO SYS_INT
..CPU_WAIT
0 0 56692 2723976 206800 734212 0 0 0 1 1 1 2 0 97 0
1 0 56692 2723651 206800 734472 0 0 0 0 2133 2887 3 1 89 7
1 0 56692 2723651 206800 734472 0 0 0 0 459 974 0 0 92 8
1 0 56692 2723651 206800 734472 0 0 0 0 2133 2887 3 1 80 11
1 0 56692 2723658 206800 734472 0 0 0 0 459 974 0 0 100 0
..
..
.....

```

Figure 35 : Résultat généré par la sonde *Vmstat*

Ce fichier de résultats sera traité, en fin de collecte, par le moteur de traitement qui transforme le contenu vers le format CBE avant de l'exporter dans un fichier en format CSV.

2.2. Autres exemples de sondes

Pour disposer d'une cartographie plus complète de l'activité d'un système, il est souvent utile de combiner la mesure de la consommation des ressources systèmes avec la mesure des débits sur les interfaces réseaux. Cette corrélation peut servir dans la prise de décision pour d'éventuelles évolutions de l'architecture du système.

Pour mesurer le débit sur une interface réseau, il existe des techniques adaptées en fonction des systèmes d'exploitation :

- Sous Windows, les compteurs de performance sont accessibles via la commande système « Perfmon » qui permet de les visualiser, en temps réel, sous forme de graphes de variation. Une autre solution consiste à développer son propre programme pour accéder à

ce type d'information. En langage Perl, on peut s'inspirer de la bibliothèque « Win32 ::Perflib » pour accéder aux compteurs de performance Windows.

- Sous Linux, une simple lecture du fichier `/proc/dev/net` donne des informations régulièrement mises à jour sur la charge des interfaces réseaux

En ce qui nous concerne, il existe la commande *ifstat*, dont l'implémentation est disponible dans de nombreux systèmes d'exploitation⁴. Cette commande s'exécute de la même façon que la commande *Vmstat* avec un argument en plus qui précise l'interface réseau à observer. La sortie de la commande *ifstat* est illustrée dans la Figure 36.

<i>Interface réseau</i>	
KB/s in	KB/s out
2.65	1.06
8.34	11.21
2.81	0.36
6.81	9.73
1.55	0.00
.....

Figure 36 : Exemple de sortie de la commande Ifstat

La corrélation entre les informations liées à l'utilisation des ressources systèmes (sortie de *Vmstat*) et celles liées à l'utilisation des ressources réseaux (sortie *Ifstat*) sont très utiles car cela permet de prévoir quelle ressource peut être la cause d'un éventuel goulot d'étranglement et d'anticiper, en conséquence, les évolutions futures de l'application ou de son environnement.

⁴ <http://gael.roualland.free.fr/ifstat/>

3. Implémentation du moteur de traitement

Le moteur de traitement est une réadaptation du module « Generic Log Adapter » intégré dans la plate-forme TPTP. La structure interne de ce module est rappelée dans la Figure 37.

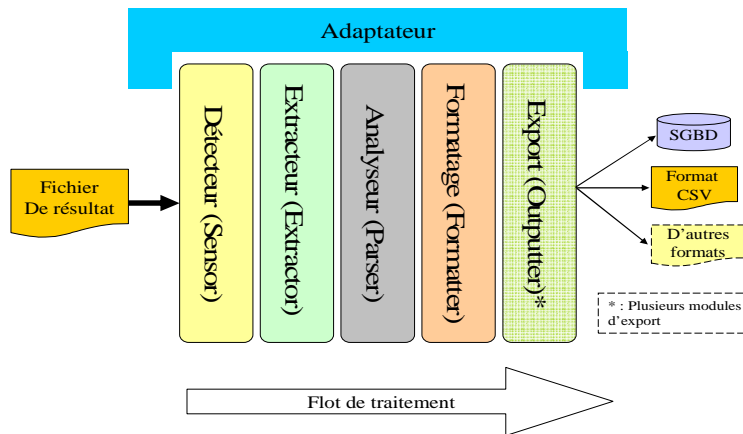


Figure 37 : Structure interne du "GLA"

Les évolutions apportées à ce module concernent les parties suivantes :

- L'analyseur « parseur »
- Le module d'export en format CSV
- L'écriture d'un adaptateur

3.1. L'analyseur « parseur »

Il existe deux techniques pour analyser un fichier à l'aide du module « GLA » :

- Une analyse basée sur les règles : adaptée pour un fichier dont le format du contenu est uniforme. Cette technique utilise des règles fixes pour délimiter les champs contenus dans le fichier. TPTP fournit un éditeur Eclipse pour faciliter la création des règles et vérifier, au fur et à mesure, qu'elles fonctionnent correctement (Cf. Figure 38).

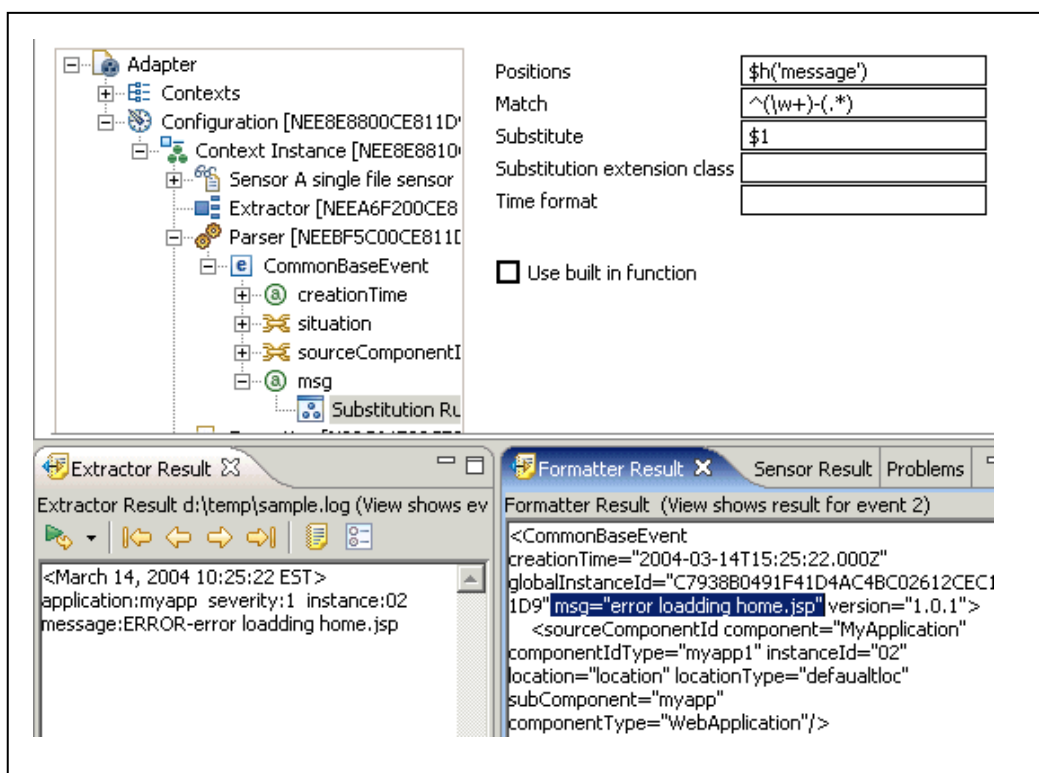


Figure 38 : Editeur Eclipse pour la création d'un « parseur »

- Une analyse statique : cette technique est utilisée pour les fichiers dont il est difficile d'appliquer la première méthode. Ceci est le cas pour le fichier de résultats produit par la sonde Vmstat et où le contenu est hétérogène. Le principe de cette solution consiste à créer une classe Java où on décrit les actions entreprises pour analyser les informations contenues dans le fichier de résultats et les transcrire dans le format CBE.

En terme de performance, la deuxième technique permet un traitement, nettement plus rapide, des fichiers de grande taille (Cf. ANNEXE H) car la classe Java qui analyse les fichiers est mieux optimisée.

3.2. Structuration des informations

Le formalisme CBE est utilisé comme format pivot dans le moteur de traitement. Ce format est décrit partiellement dans l'ANNEXE F. Il est extensible et permet d'ajouter de nouveaux champs grâce notamment à la propriété « **ExtendedDataElements** ». Cette dernière nous sera utile pour inscrire les diverses métriques collectées par la sonde Vmstat (mémoire vive, processeur utilisateur, etc.)

Finalement, les propriétés du modèle CBE, qui seront utilisées pour représenter les mesures collectées, sont les suivantes:

- **CreationTime** : l'horodatage de la mesure

- **Location** : le nom de la machine sur laquelle sont déployées les sondes.
- **ExtendedDataElements** : cette propriété sera utilisée autant de fois qu'il y a de métriques différentes. Pour l'exemple de la sonde *Vmstat*, on dispose de seize métriques différentes.

La Figure 39 représente la réécriture du fichier de résultats, généré par la sonde, et ceci en utilisant le formalisme CBE.

```

<CommonBaseEvent creationTime="2005-02-04T16:31:59.000000" version="1.0.1">
  <extendedDataElements name="PROCS_R" type="string"><values>0</values></extendedDataElements>
  <extendedDataElements name="PROCS_B" type="string"><values>0</values></extendedDataElements>
  <extendedDataElements name="MEM_SWAP" type="string"><values>56992</values></extendedDataElements>
  <extendedDataElements name="MEM_FREE" type="string"><values>65.84</values></extendedDataElements>
  <extendedDataElements name="MEM_BUF" type="string"><values>206800</values></extendedDataElements>
  ...
  <sourceComponentId .executionEnvironment="LINUX" location="192.168.10.117" componentType="VMSTAT" />
</CommonBaseEvent>
<CommonBaseEvent creationTime="2005-02-04T16:32:00.000000" version="1.0.1">
  <extendedDataElements name="PROCS_R" type="string"><values>1</values></extendedDataElements>
  <extendedDataElements name="PROCS_B" type="string"><values>0</values></extendedDataElements>
  <extendedDataElements name="MEM_SWAP" type="string"><values>56992</values></extendedDataElements>
  <extendedDataElements name="MEM_FREE" type="string"><values>65.84</values></extendedDataElements>
  <extendedDataElements name="MEM_BUF" type="string"><values>206800</values></extendedDataElements>
  ...
  <sourceComponentId .executionEnvironment="LINUX" location="192.168.10.117" componentType="VMSTAT" />
</CommonBaseEvent>
.
.
.
<CommonBaseEvent creationTime="2005-02-04T16:32:02.000000" version="1.0.1">
  <extendedDataElements name="PROCS_R" type="string"><values>1</values></extendedDataElements>
  <extendedDataElements name="PROCS_B" type="string"><values>0</values></extendedDataElements>
  <extendedDataElements name="MEM_SWAP" type="string"><values>56992</values></extendedDataElements>
  <extendedDataElements name="MEM_FREE" type="string"><values>65.84</values></extendedDataElements>
  <extendedDataElements name="MEM_BUF" type="string"><values>206800</values></extendedDataElements>
  ...
  <extendedDataElements name="CPU_USER" type="string"><values>0</values></extendedDataElements>
  <extendedDataElements name="CPU_SYS" type="string"><values>1</values></extendedDataElements>
  <extendedDataElements name="MEM_IDL" type="string"><values>99</values></extendedDataElements>
  <sourceComponentId .executionEnvironment="LINUX" location="192.168.10.117" componentType="VMSTAT" />
</CommonBaseEvent>

```

Figure 39 : Structuration des résultats dans le format CBE

3.3. Le module d'export « outputter »

Le module qui a été développé permet d'exporter les informations structurées en format CBE dans un fichier CSV. Les modules d'exports qui sont fournis par défaut dans TPTP permettent d'exporter ces informations dans des fichiers XML. Pour ajouter ce nouveau module d'export, il fallait développer une classe qui étend la classe

« **org.eclipse.hyades.logging.adapter.impl.ProcessUnit** » et implémente l'interface

« **org.eclipse.hyades.logging.adapter.IOutputter** ».

A l'issue de cette opération d'export, nous obtenons le fichier au format CSV suivant :

```
Date;Host;PROCS_R;PROCS_B;MEM_SWAP;MEM_FREE;MEM_BUF;MEM_CACH;SWAP_SI;..;SR;CPU_SYS;CPU_IDL;CPU_WAIT
2007-02-04 16:31:59;192.168.10.117;0;0;56692;2723976;206800;734212;0;0;0;1;1;1;2;0;97;0
2007-02-04 16:31:59;192.168.10.117;1;0;56692;2723651;206800;734472;0;0;0;0;2133;2887;3;1;89;7
2007-02-04 16:31:59;192.168.10.117;1;0;56692;2723651;206800;734472;0;0;0;0;459;974;0;0;92;8
2007-02-04 16:31:59;192.168.10.117;1;0;56692;2723651;206800;734472;0;0;0;0;2133;2887;3;7 ;80;10
2007-02-04 16:31:59;192.168.10.117;1;0;56692;2723658;206800;734472;0;0;0;0;459;974;0;0;100;0
..
..
..
```

Figure 40 : Fichier de résultats Vmstat au format CSV

Ce fichier peut être traité directement par des tableurs pour générer des graphiques. Nous avons développé un module de présentation qui génère automatiquement des graphiques présentant l'utilisation du processeur et de la mémoire dans le temps.

3.4. Création de l'adaptateur

Le fichier adaptateur (extension du fichier = « adapter ») est un fichier *XML* qui décrit la configuration du moteur de traitement. On y déclare les paramètres et les classes java utilisées au niveau de chaque sous-module de ce moteur. Chaque sonde est associée à un « template » qui est un fichier adaptateur générique avec un jeu de variables. Ce « template » est instancié dans la phase d'initialisation, au début de la supervision, et les variables qu'il contient sont substituées par les données saisies par l'utilisateur (Cf. Figure 41)

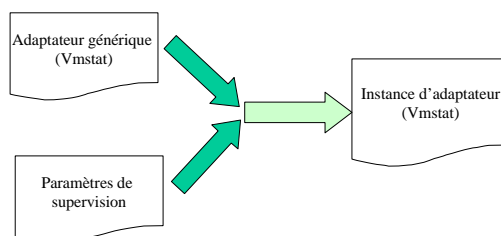


Figure 41 : Instanciation d'un adaptateur

Le fichier générique de l'adaptateur pour la sonde *Vmstat* est décrit dans la figure suivante

```

<?xml version="1.0" encoding="UTF-8"?>
<adapter:Adapter xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
.....
  Context.xsd http://www.eclipse.org/hyades/schema/ComponentConfiguration.xsd
  ComponentConfiguration.xsd http://www.eclipse.org/hyades/schema/Sensor.xsd Sensor.xsd">

  <hga:Contexts>
    <hga:Context description="Context Instance for the current component"
      executableClass="org.eclipse.hyades.logging.adapter.impl.BasicContext"
      implementationCreationDate="2003-11-25T13:40:00.0" implementationVersion="1.0.0"
      loggingLevel="50" name="Basic Context Implementation" role="context"
      roleCreationDate="2003-11-25T13:40:00.0"
      roleVersion="1.0.0" uniqueID="StaticParserContextID1">
      <hga:Component description="Static Parser Sensor"
        executableClass="org.eclipse.hyades.logging.parsers.adapter.sensors.StaticParserSensor"
        implementationCreationDate="2003-11-25T13:40:00.0" implementationVersion="1.0.0"
        loggingLevel="50" name="Static Parser Sensor" role="sensor"
        roleCreationDate="2003-11-25T13:40:00.0" roleVersion="1.0.0" uniqueID="sensorID1"/>
      <hga:Component description="Log File Outputter"
        executableClass="com.kereval.tptp.outputters.CsvOutputter"
        implementationCreationDate="2005-02-22T22:10:51.0" implementationVersion="1.0"
        loggingLevel="30" name="JVMSTAT" role="outputter" roleVersion="1.0" uniqueID="N93CEE3B5"/>
    </hga:Context>
  </hga:Contexts>

  <cc:Configuration description="The component level configurations for this Adapter"
    uniqueID="StaticParserConfigurationID1">
    <cc:ContextInstance continuousOperation="false" description="Context Instance for the current component"
      maximumIdleTime="20000" pauseInterval="10" uniqueID="StaticParserContextID1">
      <cc:Sensor description="A static parser sensor" uniqueID="sensorID1" maximumBlocking="30"
        type="StaticParserSensor">
        <pu:Property propertyName="directory" propertyValue="@srcRepository@"/>
        <pu:Property propertyName="fileName" propertyValue="@srcFile@"/>
        <pu:Property propertyName="parserClassName" propertyValue="com.kereval.tptp.parser.StaticParserVmstat"/>
        <pu:Property propertyName="delayTimeStamp" propertyValue="@delay@"/>
      </cc:Sensor>
      <cc:Outputter description="Kereval CSV Outputter" uniqueID="N93CEE3B5" type="undeclared">
        <pu:Property propertyName="fileName" propertyValue="@destFile@"/>
        <pu:Property propertyName="directory" propertyValue="@destRepository@"/>
      </cc:Outputter>
    </cc:ContextInstance>
  </cc:Configuration>
</adapter:Adapter>

```

Figure 42: Adaptateur générique pour la sonde Vmstat

Les lignes en caractères gras sont les variables qui seront substituées par leurs valeurs. Ces variables sont :

- @srcRepository@ : répertoire du fichier de résultats
- @srcFile@ : le nom du fichier de résultats (fichier généré par la sonde)
- @destRepository@ : le répertoire destination pour le fichier généré (exporté) par le module de traitement
- @destFile@ : le nom du fichier généré par le module de traitement (le fichier au format CSV)
- @delay@ : le décalage temporel entre la machine supervisée et la machine de référence⁵

⁵ Rappel : c'est la machine où est déployée le module de traitement (machine superviseur)

4. Fonctionnement de la plate-forme

Le fonctionnement de la plate-forme de supervision est régi par un ensemble de tâches Ant. Ce choix de développement facilite l'automatisation des activités de la supervision. De plus, cette solution est portable et permet une meilleure structuration et organisation du projet.

Ces tâches sont au nombre de quatre et concernent le transfert des fichiers, le pilotage des sondes, le traitement des fichiers de résultat et finalement la tâche de présentation « Reporting ».

4.1. Tâche de transfert des fichiers

Cette tâche est exécutée lors du déploiement des sondes et de la restitution des fichiers de résultats. Le transfert des fichiers peut être local ou distant en fonction de leurs emplacements par rapport à la machine superviseur :

❖ Transfert local

Le transfert local des fichiers s'appuie sur la commande *copy*, du framework Ant. Cette commande s'exécute avec la syntaxe suivante :

```
<target name="taskCopy">
    <copy file="${srcFile}" todir="${destDir}"/>
</target>
```

Figure 43 : Tâche Ant pour le transfert local des fichiers

Les variables `${srcFile}` et `${destDir}` correspondent respectivement au chemin absolu pour le fichier source et le répertoire de destination.

❖ Transfert distant

Le transfert distant des fichiers nécessite les paramètres de connexion à la machine distante. Sur cette machine, il faut déployer un serveur SSH (exemple *OpenSSH*⁶ pour Linux et *sshwindows*⁷ pour Windows) afin de permettre un transfert sécurisé des fichiers. Nous utilisons le paquetage PuTTY⁸ et plus particulièrement l'utilitaire *PSCP*⁹ (*PuTTY Secure CoPy*) pour réaliser ce transfert. La syntaxe d'utilisation de cet utilitaire est la suivante:

```
pscp -pw <password> <srcFile> <login>@<remoteHost>:<destDir> ① (put)
pscp -pw <password> <login>@<remoteHost>:<srcFile> <destDir> ② (get)
```

Figure 44 : Syntaxe d'utilisation de la commande pscp

⁶ <http://www.openssh.com/>

⁷ <http://sshwindows.sourceforge.net/>

⁸ <http://www.chiark.greenend.org.uk/~sgtatham/putty/>

⁹ PSCP est une adaptation de Secure Copy (SCP) d'UNIX pour Ms-DOS

Suivant le type de transfert, vers ou depuis la machine distante, l'emplacement des variables `<destDir>` et `<srcFile>` peut être inversé (Figure 44). Pour différencier les deux cas, nous utilisons une variable « mode » qui peut prendre l'une des deux valeurs « put » ou « get ».

Finalement, la tâche de transfert distant est déclarée de la manière suivante :

```
<target name="taskCopy">
  <taskRemoteSecureCopy mode="${mode}" srcFile="${srcFile}"
    destDir="${destDir}" host="${host}" user="${user}"
    password="${password}" />
</target>
```

Figure 45 : Tâche Ant pour le transfert distant des fichiers

L'entrée « **taskRemoteSecureCopy** » correspond à la déclaration d'une nouvelle tâche Ant associée à une classe Java. Cette classe doit étendre la classe `org.apache.tools.ant.Task` de Ant et doit contenir la méthode `execute()` qui représente l'équivalent de la méthode « main » qu'on trouve dans une classe Java classique.

4.2. Tâche de pilotage des sondes

Cette tâche concerne le démarrage et l'arrêt d'une sonde et nous nous intéressons uniquement au cas d'une sonde distante. Pour exécuter une commande sur une machine distante il existe plusieurs solutions (RMI¹⁰, RPC¹¹, etc.) mais celle qui nous semble la plus simple à mettre en œuvre consiste à utiliser l'utilitaire « *plink* » disponible dans le paquetage de « PuTTY ». « *Plink* » est un client SSH qui permet d'ouvrir une session sécurisée sur une machine distante. La syntaxe de « *plink* » est la suivante :

```
plink -pw <password> <login>@<remoteHost> "<commande>"
```

Figure 46 : Syntaxe d'utilisation de la commande *plink*

L'argument `<commande>` sera substitué par la commande qui permet l'exécution ou l'arrêt d'une sonde. Dans un environnement Unix, nous utilisons la commande `ssh` qui rend le même service que l'utilitaire *plink* et s'exécute avec la même syntaxe.

¹⁰ Remote Method Invocation

¹¹ Remote Procedure Call

La tâche Ant qui permet de piloter la sonde Vmstat est déclarée de la manière suivante:

```
<target name="taskVmstat">
  <taskVmstat mode="${mode}" frequency="${vmstat.sup.frequency}"
    count="${vmstat.sup.count}" outputFile="${vmstat.sup.log}"
    scriptVmstat="${vmstat.script.name}" repositoryResult="${vmstat.sup.repository}"
    host="${vmstat.sup.host}" user="${vmstat.sup.user}" password="${vmstat.sup.password}"/>
</target>
```

Figure 47 : Tâche Ant pour le pilotage de la sonde Vmstat

Les valeurs des arguments nécessaires pour exécuter la sonde *Vmstat* sont déclarées dans un fichier de propriétés dont un extrait est donné dans la Figure 48

```
vmstat.sup.frequency = la fréquence de collecte
vmstat.sup.count = le nombre de répétition de la collecte
vmstat.sup.log = le nom de fichier généré par vmstat
vmstat.sup.name = le nom du script Vmstat (nom de la sonde)
vmstat.sup.repository = répertoire cible pour le fichier de résultats généré
vmstat.sup.host = la machine hébergeant le SUT (nom_machine ou @IP)
vmstat.sup.user = un compte utilisateur valide pour se connecter
vmstat.sup.password = le mot de passe de l'utilisateur
```

Figure 48 : Déclaration des paramètres la sonde Vmstat

A l'instar de la tâche « *taskRemoteCopy* », il faut créer une classe Java associée à la tâche « *taskVmstat* » définie dans la Figure 47. Cette classe doit étendre aussi la classe *org.apache.tools.ant.Task* de Ant et doit contenir la méthode *execute()*.

4.3. Tâche de traitement des fichiers de résultat

Le traitement des fichiers de résultat est assuré par le moteur d'analyse basée sur le module « GLA » de la plate-forme TPTP. Ce dernier existe sous forme d'un plug-in Eclipse intégré dans TPTP et sous forme d'un module « *stand-alone* » qui peut être exécuté en ligne de commande. Nous utilisons la deuxième option qui nous permet de démarrer le moteur d'analyse à partir d'une tâche Ant de la façon décrite dans la Figure 49.

```

<target name="analyserLog">
  <property name="adapter" value="${fileAdapter}" />
  <java classname="org.eclipse.hyades.logging.adapter.Adapter" fork="true">
    <arg line="-cc ${adapter} -ac ${adapter}" />
    <jvmarg value="-DGLA_HOME=${gla.home}" />
    <classpath>
      <fileset dir="${gla.home}" includes="lib/*.jar"/>
    </classpath>
  </java>
</target>

```

Figure 49 : Tâche Ant pour l'analyse des fichiers

Les variable `${adapter}` et `${gla.home}` correspondent respectivement au chemin absolu vers le fichier adaptateur et à l'emplacement des bibliothèques java (les fichiers jar) ; indispensables pour le fonctionnement du moteur d'analyse.

4.4. Tâche de reporting

La tâche de reporting se charge de générer des graphiques à partir des fichiers de résultat au format CSV. La génération des graphes s'appuie sur la bibliothèque java *JfreeChart*¹² qui permet de créer des graphes de façon automatique et de les exporter sous la forme d'image. La Figure 50 est un exemple de tels graphes.

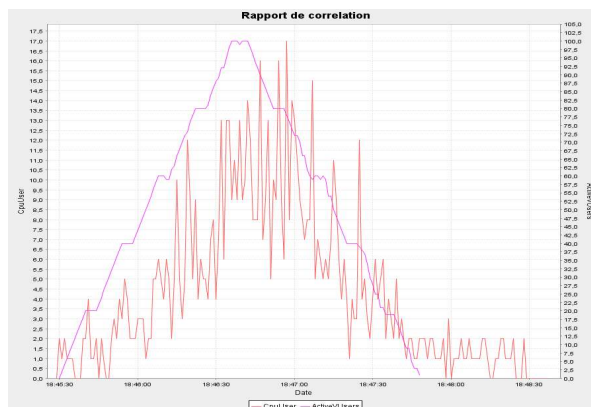


Figure 50 : Exemple de graphique généré avec la bibliothèque JfreeChart

La tâche Ant qui gère le reporting est déclarée de la façon suivante :

```

<target name="taskGraphCorrelate">
  <taskGraphCorrelate argument="${arg}" outputImg="${outputImg}" chartTitle="${headerTitle}" />
</target>

```

Figure 51 : Tâche Ant pour la génération et la corrélation des graphiques

¹² <http://www.jfree.org/jfreechart/>

Dans la variable *argument* on renseigne les fichiers et les métriques à représenter. La valeur de cette variable doit avoir la forme suivante «*file1,title1, index1@ file2,title2,..@ fileN,titleN, indexN*»

- File : chemin absolu vers le fichier de résultat
- Title : label de la métrique à tracer
- Index : index de la colonne associée à la métrique

Plusieurs métriques peuvent être tracées en les séparant par des « @ ».

5. Conclusion

Ce chapitre décrit la mise en œuvre des différentes composantes de la plate-forme de supervision et en particulier les sondes et le moteur d'analyse: nous avons implémenté une sonde permettant de collecter les informations relatives à la consommation des ressources systèmes dans un environnement Linux. Ce prototype de sonde peut servir d'exemple pour implémenter d'autres exemples (ex sonde *iostat* pour superviser les entrées/sorties)

L'implémentation du moteur d'analyse est basée sur l'outil « GLA » qui offre toute la modularité nécessaire pour prendre en charge des fichiers hétérogènes.

La plate-forme de supervision est structurée sous forme de tâches Ant ce qui facilite son pilotage par des outils externes. A chaque fonctionnalité de la supervision est associée une tâche Ant. On dispose alors de:

- Une tâche pour le déploiement des sondes et utilisée également pour la récupération des résultats
- Une tâche pour le pilotage des sondes
- Une tâche pour l'analyse des fichiers de résultat
- Une tâche pour la génération des graphiques

La dernière partie de ce rapport est consacrée à valider expérimentalement notre solution de supervision et à comparer ses performances par rapport à d'autres plates-formes ou standards de supervision et notamment la plate-forme TPTP et SNMP. Au travers cette comparaison, on a démontré que l'approche de supervision « *off-line* » est moins intrusive, et présente d'autres avantages comme le passage à l'échelle.

PARTIE 4
EXPERIMENTATIONS

F. Validation expérimentale

Ce chapitre a pour objectif de comparer trois solutions de supervision : la plate-forme TPTP, le standard SNMP et notre plate-forme de supervision. Il s'agit dans cette comparaison de confronter trois approches de supervision : deux approches « *online* », celles de TPTP et de SNMP, qui s'appuient toutes les deux, sur des agents installés sur le SUT et qui utilisent un mode de communication de type client/serveur. La troisième approche « *offline* » correspond à notre plate-forme de supervision : elle utilise des sondes pour collecter les informations liées à l'activité du système.

Le critère principal de comparaison est celui de l'intrusivité avec ces deux variantes :

- a) - L'intrusivité au niveau de la consommation des ressources système : nous mesurons l'*overhead* généré par chacune de ces trois solutions. Cette mesure consiste à évaluer l'impact de la supervision en terme de temps d'exécution d'un script étalon développé en PHP.
- b) - L'intrusivité au niveau de l'utilisation des ressources réseaux : nous analysons les paquets générés par TPTP et SNMP à l'aide de l'outil Wireshark¹ (anciennement Ethereal) qui est un analyseur de protocole réseau.

1. Description du contexte de l'expérimentation

1.1. Plate-forme expérimentale

L'environnement d'expérimentation est composé de deux machines connectées entre elles via un réseau à haut débit. La configuration et les caractéristiques de ces machines sont décrites dans la Figure 52

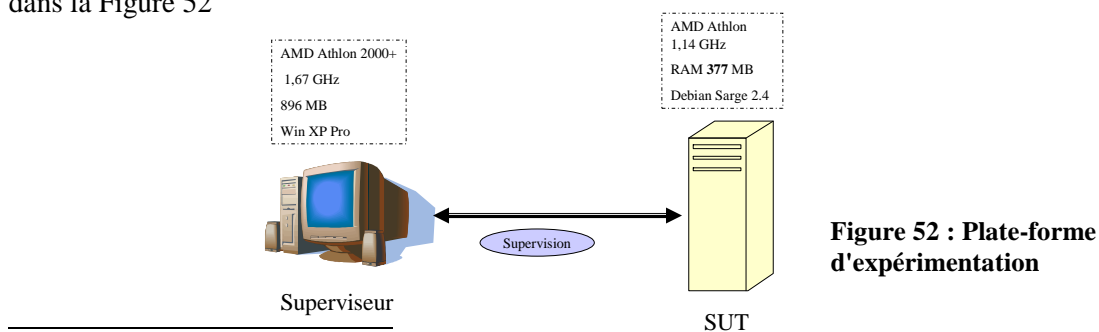


Figure 52 : Plate-forme d'expérimentation

¹ <http://www.wireshark.org/>

Sur la machine SUT est déployé un agent SNMP, un contrôleur d'agent (RAC) TPTP et la sonde Vmstat que nous avons développée en Script Shell. Le Tableau 3 dresse la liste des versions des autres modules installés sur le SUT pour le besoin de l'évaluation.

Module	Version
PHP	4.4.4-8
PEAR	1.6.1
BENCHMARK	1.2.7
Remote Agent	4.4.0.1
SNMP	5.1.2-6.2
JAVA	1.5.0_06

Tableau 3 : Liste des versions des modules installés sur la machine SUT

Sur la machine superviseur, on installe un client SNMP qui s'appuie sur la bibliothèque java **snmp4j**². La version d'Eclipse utilisée est 3.1.2 et la version de TPTP mise en œuvre pour notre évaluation est la 4.4.0.1 qui est la plus récente lors de l'expérimentation.

1.2. Démarche expérimentale

Notre démarche expérimentale s'intéresse particulièrement aux deux aspects de l'intrusivité : l'intrusivité au niveau des ressources systèmes et l'intrusivité au niveau du réseau.

Le premier aspect de l'intrusivité n'est pas une caractéristique d'un outil de supervision mais dépend de l'environnement expérimental. Il est difficile à quantifier en se basant uniquement sur des mesures singulières de la quantité des ressources systèmes qu'il requière. Néanmoins, il est communément admis qu'un outil est considéré moins intrusif qu'un autre s'il engendre une faible augmentation du temps d'exécution de l'application supervisée. Partant de cette hypothèse, nous avons développé un script en langage PHP, que nous avons déployé sur la machine SUT. Le contenu de ce script est donné dans la Figure 53.

² <http://www.snmp4j.org/>

```

<?
require_once 'Benchmark/Timer.php';
$timer = new Benchmark_Timer();
$stab= array();
$timer->start();
for ($i=0; $i<434000; $i++)
{
    $stab[$i]='1234567890';
}
$Total=0;
for ($i=0; $i<40000000; $i++)
{
    $Total +=$i;
}
$timer->setMarker('Marker 1');
$timer->stop();
$timer->display();
$profiling = $timer->getProfiling();
?>

```

Figure 53 : Script PHP utilisé pour l'intrusivité

Ce script a été conçu pour permettre la surcharge des ressources systèmes (processeur et mémoire) sur le système SUT. L'objectif escompté est de rendre ce dernier plus sensible à une perturbation comme celle engendrée par l'activité de la supervision. La saturation de la mémoire est réalisée en remplissant plusieurs tableaux et la saturation du processeur passe par l'utilisation d'une boucle suffisamment grande pour occuper le système. Ce script a le mérite d'être simple et facile à maintenir et à évoluer pour surcharger d'autres ressources systèmes.

Pour mesurer le temps d'exécution du script PHP, il existe le module « Benchmark³ » disponible dans les bibliothèques « PEAR⁴ » de PHP, et qui permet entre autres de profiler des applications en PHP. Les mesures des temps d'exécution ont été réalisées d'abord sans l'activité de supervision et ensuite avec la supervision à l'aide de chacun des trois outils.

Le deuxième aspect de l'intrusivité (intrusivité au niveau du réseau) est évalué en mesurant le volume de trafic généré par l'activité de supervision. Nous utilisons pour cela l'outil « Wireshark » qui permet une analyse protocolaire avec mesure des tailles des paquets réseaux. Cette analyse concerne uniquement les deux outils TPTP et SNMP vue que notre approche de supervision ne génère pas de trafic réseaux au cours de la supervision. Les informations collectées avec notre approche « *offline* » sont stockées localement et ne sont transmises au superviseur qu'à la fin de la supervision.

³ <http://pear.php.net/package/Benchmark>

⁴ <http://pear.php.net/>

2. Bilan de l'expérimentation

Nous allons présenter au cours de ce paragraphe les mesures obtenues concernant l'évaluation de l'intrusivité de notre approche « *offline* » de supervision vis-à-vis d'une approche « *online* » incarnée par les deux outils SNMP et TPTP.

Nous avons multiplié le nombre des mesures réalisées pour s'assurer de la fiabilité des résultats obtenus. En effet, il est difficile de prétendre que le SUT se comportera de la même façon et que l'on obtiendra exactement les mêmes mesures sur le temps d'exécution. Il existe des processus systèmes dont il sera difficile de maîtriser l'activité comme la « journalisation ». A cela, s'ajoute le processus de gestion de la mémoire tampon « buffers » étant donné que les résultats sont redirigés vers un fichier pour une analyse postérieure.

2.1. Intrusivité au niveau des ressources systèmes

2.1.1. Résultats

Les résultats obtenus sont présentés sous forme de tableau (Tableau 4).

	Sans	Notre approche	SNMP	TPTP
Temps d'exécution (sec)	14,58	14,67	14,71	14,8
	14,59	14,7	14,76	14,82
	14,6	14,71	14,77	14,82
	14,61	14,74	14,78	14,83
	14,61	14,74	14,78	14,83
	14,61	14,75	14,79	14,83
	14,61	14,75	14,8	14,83
	14,62	14,76	14,81	14,83
	14,64	14,77	14,81	14,84
	14,65	14,77	14,82	14,84
	14,66	14,79	14,82	14,84
	14,66	14,8	14,83	14,84
	14,66	14,81	14,83	14,85
	14,66	14,81	14,83	14,85
	14,67	14,81	14,83	14,86
	14,68	14,81	14,84	14,86
	14,68	14,81	14,84	14,86
	14,68	14,82	14,84	14,86
	14,69	14,82	14,84	14,86
	14,69	14,82	14,85	14,86
	14,69	14,82	14,85	14,86
	14,7	14,83	14,85	14,9
	14,71	14,84	14,85	14,87
14,71	14,84	14,86	14,87	
14,72	14,84	14,86	14,87	
Meilleur score	14,58	14,67	14,71	14,80

Tableau 4 : Mesures des temps d'exécution

Ces résultats ont été représentés sur un graphique « radar » (Figure 54) pour une lecture plus aisée. Chaque rayon du graphique correspond à un palier des temps d'exécution du script.

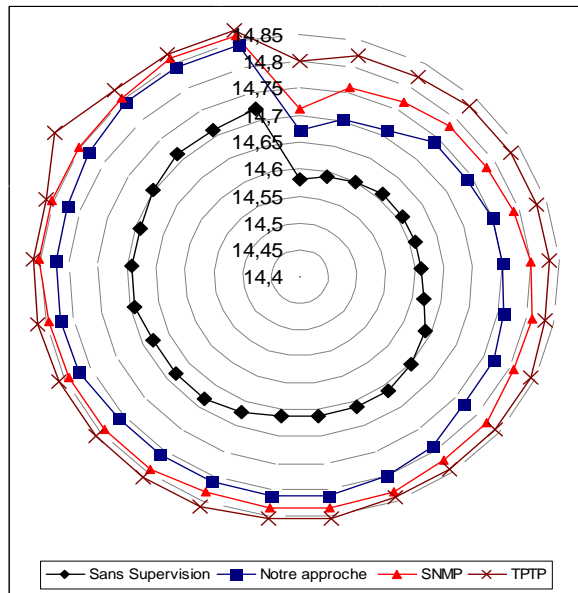


Figure 54 : Représentation « radar » des temps d'exécutions

2.1.2. Analyses

L'analyse des courbes de la Figure 54 permet d'une part de consolider les hypothèses, avancées au début de ce manuscrit, selon lesquelles notre approche de supervision « *offline* » serait moins intrusive puisqu'elle engendre un temps d'exécution supplémentaire inférieur à celui engendré par les deux autres techniques. D'autre part, la courbe représentant le cas « sans supervision » permet de confirmer que les conditions expérimentales ne sont pas idéales car on obtiendrait à la place une courbe parfaitement circulaire. Le système ne se comporte pas de manière identique et cela ne s'explique que par l'activité interne au système d'exploitation, qui ne peut être contrôlée par supervision. La forme déformée de cette courbe s'explique donc par l'existence de facteurs (ex processus) dont le fonctionnement n'est pas uniforme et qui altèrent les mesures. Cela nous amène à considérer, pour chacun des quatre cas étudiés, le meilleur score obtenu (c.à.d. temps minimum d'exécution) comme mesure de référence car elle correspond à l'influence la plus faible des facteurs parasites.

Pour calculer « l'overhead », nous utilisons la formule donnée dans l'Équation 3.

$$Overhead_{d'une\ technique} = \frac{\text{Temps}_{Technique} - \text{Temps}_{Sans\ supervision}}{\text{Temps}_{Sans\ supervision}} \times 100$$

Équation 3 : Calcul de l'overhead

Le gain relatif correspond à la diminution, en terme de temps d'exécution, apportée par notre approche par rapport aux deux autres techniques de supervision. Sa formule est donnée respectivement pour SNMP et TPTP par l'Équation 4 et l'Équation 5.

$$Gain\ relatif_{SNMP} = \frac{Temps_{SNMP} - Temps_{Notre\ approche}}{Temps_{SNMP} - Temps_{Sans\ supervision}} \times 100$$

Équation 4 : Calcul du gain relatif par rapport à SNMP

$$Gain\ relatif_{TPTP} = \frac{Temps_{TPTP} - Temps_{Notre\ approche}}{Temps_{TPTP} - Temps_{Sans\ supervision}} \times 100$$

Équation 5 : Calcul du gain relatif par rapport à TPTP

Dans le Tableau 5, on donne les valeurs de « l'overhead » et du gain relatif calculées par les formules précédentes.

	Notre approche	SNMP	TPTP
Overhead	0,6 %	0,9%	1,5%
Gain relatif de notre approche par rapport à	NON DEFINI	30 %	59%

Tableau 5 : Calcul des « overhead » et des gains relatifs

Les gains relatifs de cette approche de supervision « offline » sont considérables, et peuvent atteindre 60 % (le cas de TPTP). Ses gains peuvent être supérieurs si les sondes sont développées dans un langage plus rapide tel que le langage C. Les résultats que nous produisons donc ici sont les gains minimaux que l'on peut espérer avec notre approche.

En plus de l'intrusivité au niveau du SUT, nous avons comparé les trois approches en terme de passage à l'échelle. Ce critère, inhérent à un outil de supervision distribué, permet de mesurer l'impact d'une supervision à grande échelle sur la machine superviseur.

2.2. Passage à l'échelle « scalability »

Le scénario de l'expérimentation consiste à augmenter progressivement (toutes les 30 secondes) le nombre d'instances de supervision jusqu'à 20 instances et à mesurer l'impact sur l'utilisation des ressources systèmes au niveau de la machine superviseur. Avec le framework TPTP il était difficile d'aller aussi loin dans l'expérimentation. On s'est donc limité à neuf instances pour ce

framework. Avec SNMP et avec notre approche nous avons pu atteindre l'objectif de 20 instances. En parallèle, nous avons choisi de surveiller un ensemble de métriques systèmes qui sont respectivement : l'utilisation de la mémoire (RAM), l'utilisation du fichier d'échange⁵, les pourcentages respectifs des temps processeur en mode système et en mode utilisateur. Les résultats de cette expérimentation sont présentés dans les figures suivantes : (Figure 55), (Figure 56) et (Figure 57).

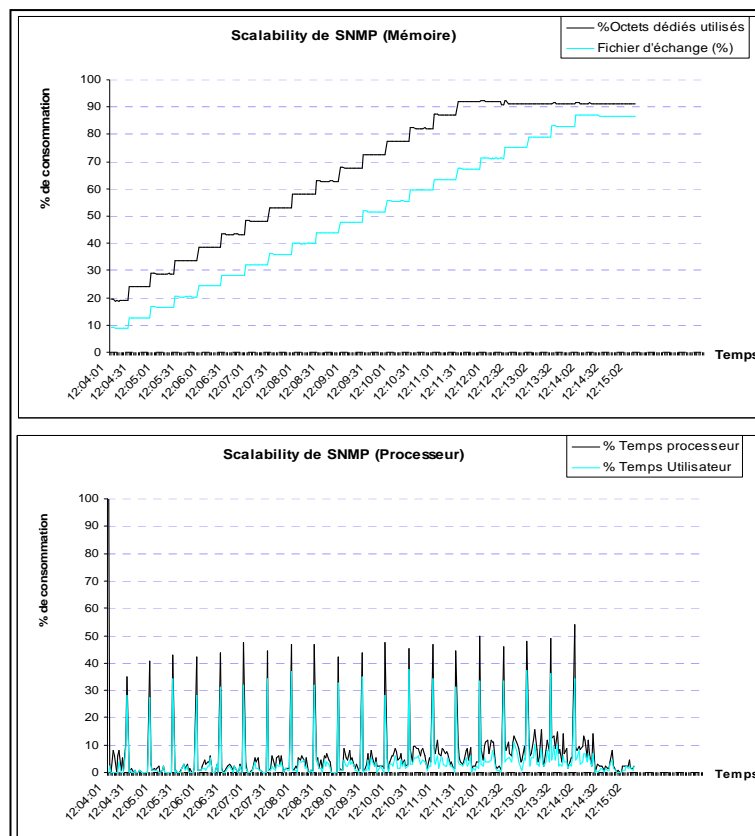


Figure 55 : Passage à l'échelle - cas du SNMP

Pour le cas de SNMP, l'utilisation de la mémoire et du fichier d'échange augmente avec le nombre d'instances de supervision⁶ (92 % pour la RAM, 87% pour les fichiers d'échanges). A partir de la quinzième instance la mémoire physique est saturée et le système continue à solliciter le fichier d'échange. Concernant l'utilisation du processeur, le même phénomène est constaté avec des pics de la consommation au moment de l'instanciation d'une nouvelle supervision.

⁵ Le fichier d'échange, *pagefile.sys*, permet de palier le manque de mémoire physique, il est créé automatiquement sur la partition où est installé votre système d'exploitation

⁶ Une instance de supervision correspond à la supervision d'une nouvelle machine

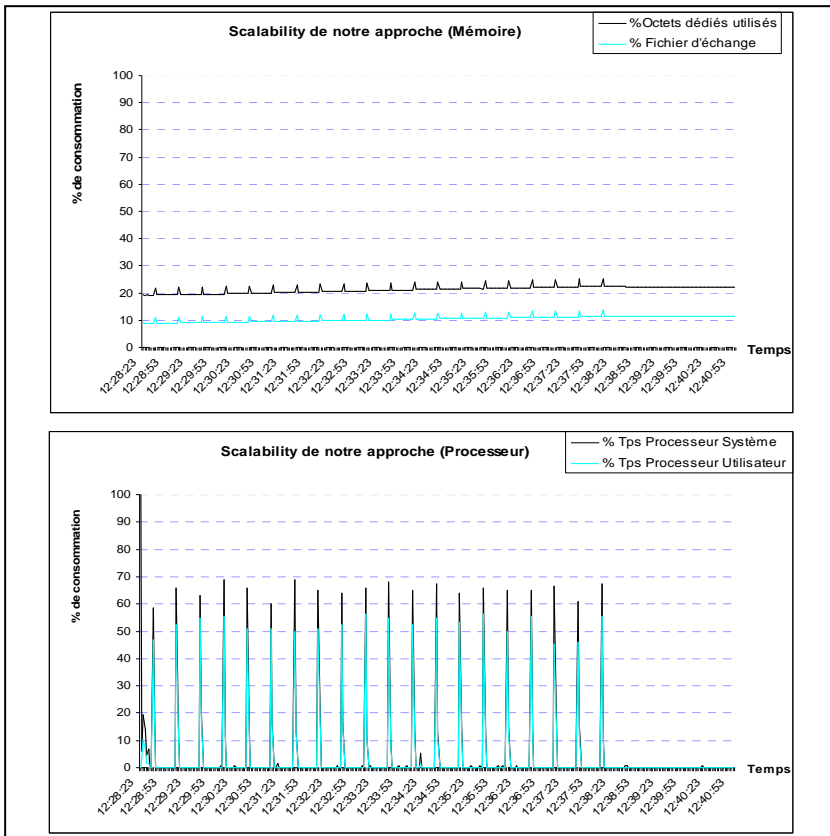


Figure 56 : Passage à l'échelle - cas de notre approche

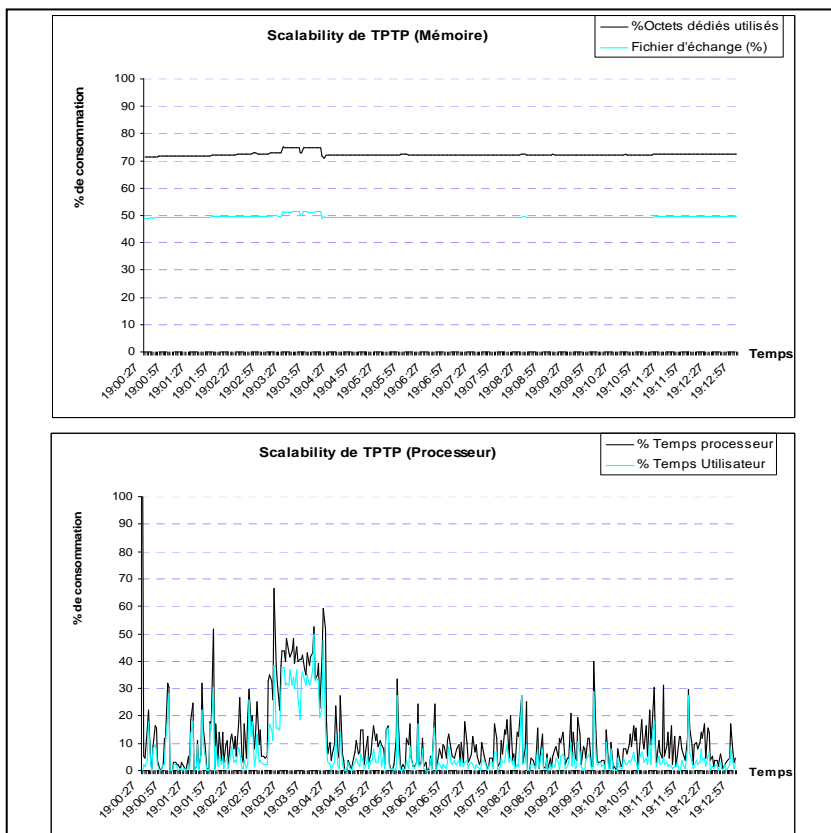


Figure 57 : Passage à l'échelle - cas de TPTP

Notre approche utilise d'une façon optimisée les ressources systèmes. En effet, la sollicitation de ces dernières n'est observée qu'au moment de l'instanciation d'une supervision ce qui correspond en réalité au déploiement de la sonde sur la machine supervisée. Après cette phase de déploiement les ressources sont libérées et la machine superviseur retrouve son état initial (c à d avant le déploiement).

Les courbes obtenues pour le cas de TPTP sont difficilement interprétables à cause de l'environnement Eclipse qui règle la consommation de la mémoire physique notamment à l'aide des options Xms et Xmx déclarées dans son fichier de configuration *eclipse.ini*.

2.2.1. Conclusions

La plate-forme TPTP est la plus intrusive. Cela s'explique par le fonctionnement de son agent contrôleur (installé sur le SUT) qui traite les informations collectées avant de les envoyer vers la machine superviseur. De plus, ce composant s'appuie sur des modules développés en Java ce qui implique la présence d'une machine virtuelle qui augmente « l'overhead » de ce framework.

Le langage de programmation de la sonde Vmstat est le Script Shell. Nous pouvons espérer réduire l' « *overhead* » de cette sonde si elle était développée dans d'autres langages tels que C, Perl ou Python réputés plus rapides. Ces langages sont des langages compilés et donc sont plus rapides que les langages interprétés tels que le Shell. Malheureusement, la contrainte du temps ne nous a pas permis d'aller plus loin dans l'expérimentation et d'évaluer le gain en performance avec des sondes développées dans l'un des langages précédemment cités.

Un autre élément important qui s'ajoute à l'intrusivité d'un framework de supervision au niveau SUT est la surcharge au niveau de la machine superviseur. Bien que cette contrainte n'ait pas d'impact direct sur le fonctionnement du SUT, elle limite néanmoins le nombre d'éléments supervisés en raison des ressources systèmes limitées sur cette machine. Les frameworks de supervision « online » tels que TPTP et SNMP présentent cette contrainte vu que les informations collectées doivent être traitées au fur et à mesure de la supervision par le superviseur. Dans notre approche, cette contrainte est nulle, puisqu'il n'y a aucune interaction entre le superviseur et le SUT durant la supervision. En d'autres termes, le superviseur n'est pas sollicité durant la supervision. Cette contrainte détermine le niveau de passage à l'échelle « scalability » que peut permettre un framework de supervision.

2.3. Intrusivité au niveau des ressources réseaux

Une plate-forme de supervision de type « *online* » se caractérise par un couplage temporel entre le superviseur et le SUT. Ceci se traduit par un échange de messages entre ces deux entités dont le volume dépend du mode et du protocole de communication qui les relie. Dans la suite

de ce paragraphe, nous détaillons ce volume pour les deux outils évalués, à savoir le framework SNMP et la plate-forme TPTP.

2.3.1. Cas TPTP

Dans la plate-forme TPTP, l'échange de messages s'effectuent entre la console Eclipse et le contrôleur d'agents (RAC). L'analyse du réseau, entre ces deux modules, avec l'outil Wireshark donne les résultats décrits dans le Tableau 6.

Couches Réseaux	Détail	Taille de la req (Octets)	Taille de la rép 1 (Octets)	Taille de la rép 2 (Octets)
Accès au réseau	Source	6	6	6
	Destination	6	6	6
	Type	2	2	2
	Total	14	14	14
IP	Version	1	1	1
	DiffServ	1	1	1
	Longueur totale	2	2	2
	Identification	2	2	2
	Frgment Offset	2	2	2
	TTL (Time to live)	1	1	1
	Protocole	1	1	1
	Entête Checksum	2	2	2
	Source	4	4	4
	Destination	4	4	4
	Total	20	20	20
Transport (TCP)	Port Source	2	2	2
	Port Destination	2	2	2
	N° Séquence	4	4	4
	N° Acquieement	4	4	4
	Longueur entête	1	1	1
	Flag	1	1	1
	Taille fenêtre	2	2	2
	Checksum	2	2	2
	Option	12	12	12
	Remplissage	2	2	2
Total	32	32	32	
Application	Charge utile		1248	173
	Total		1248	173
Total		66	1314	239

Tableau 6 : Détail des paquets échangés dans la plate-forme TPTP

La requête (colonne 3 à partir de la gauche) correspond au message envoyé par la console Eclipse vers le contrôleur d'agent. Il s'agit d'un acquittement [ACK], propre au protocole TCP, dont la taille (66 octets) est fixe et ne dépend pas du nombre des métriques collectées sur le SUT (16 dans notre cas).

Les deux colonnes (1 et 2 à partir de la droite) correspondent à la réponse du RAC contenant les informations de supervision. La taille de cette réponse (1314+239 = 1553 octets) comprend une partie à taille fixe qui sont les entêtes réseaux (66 octets) et une partie variable qui est la charge utile. La Figure 58 représente un extrait des données TPTP contenant les informations de supervision

```

<ContiguousObservation memberDescriptor="C10017" time="1184769081145" value="0.7" />
<ContiguousObservation memberDescriptor="C10018" time="1184769081145" value="0.5" />
<ContiguousObservation memberDescriptor="C11022" time="1184769081145" value="138.00" />

```

Figure 58 : Description des données applicatives de TPTP

Le langage XML est utilisé comme formalisme d'échange. La balise «ContiguousObservation» permet de délimiter les informations de supervision collectées. Chacune de ces informations est complètement définie par les champs suivants :

- ❖ *memberDescriptor* : l'identifiant unique de la métrique collecté (mémoire cache, CPU user, etc.)
- ❖ *time* : l'heure actuelle « timestamp » sur le SUT au moment de la collecte de l'information,
- ❖ *value* : la valeur la métrique collectée.

En moyenne, une métrique mesurée avec le framework TPTP occupe une taille d'environ 89 octets de la charge utile (niveau application de la pile réseau). On obtient alors pour les seize métriques que nous collectons une taille d'environ 1424 octets (une approximation de la somme des deux valeurs 1248 et 173).

En résumé, les données applicatives sont absentes dans les messages envoyés par la console Eclipse vers l'agent contrôleur (RAC). Ces messages ne sont que des acquittements TCP dont la taille vaut 66 octets. Les messages envoyés par le RAC, et qui contiennent les informations de supervision collectées sur le SUT, ont une taille variable qui est fonction du nombre de métriques collectées. Elle vaut approximativement :

$$\begin{aligned}
 \text{Taille}_{\text{réponse}} &= \text{trame TCP} + \text{charge utile (en octets)} \\
 &\sim 66 + (89 \times N) \text{ ou } N \text{ est le nombre de métrique} \\
 \\
 \text{Taille}_{\text{requête}} &= \text{trame TCP d'acquiescement} \\
 &\sim 66 \text{ octets} \\
 \\
 \text{Taille}_{\text{requête/réponse}} &= 132 + (89 \times N)
 \end{aligned}$$

Figure 59 : Formule approximative pour le calcul d'une réponse TPTP

Cette formule ne tient pas compte des traitements affectés par le médium d'accès au réseau comme le processus de fragmentation/ réassemblage. La Figure 60 donne une représentation graphique du volume du trafic généré par la plate-forme TPTP.

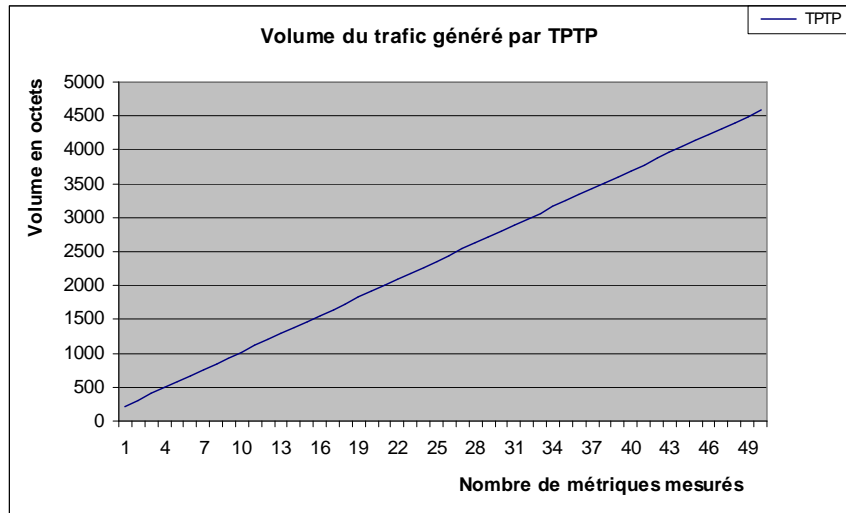


Figure 60 : Volume de trafic généré par TPTP

2.3.2. Cas SNMP

Dans un échange SNMP, le mode de transport utilisé est UDP (mode non connecté) ce qui réduit le volume des échanges entre le client et l'agent SNMP comparé à un échange en mode connecté s'appuyant sur TCP. Le mode de communication entre le client et l'agent est de type client/serveur. L'analyse du trafic SNMP avec l'outil Wireshark a permis d'identifier la structure des messages SNMP. Ces résultats sont décrits dans le Tableau 7.

Couches Réseaux	Détail	Taille de la req (Octets)	Taille de la rép (Octets)
Accès au réseau	Source	6	6
	Sestination	6	6
	Type	2	2
	Total	14	14
IP	Version	1	1
	DiffServ	1	1
	Longueur totale	2	2
	Identification	2	2
	Frgament Offset	2	2
	TTL (Time to live)	1	1
	Protocole	1	1
	Entête Checksum	2	2
	Source	4	4
	Destination	4	4
Total	20	20	
Transport (UDP)	Port Source	2	2
	Port Destination	2	2
	Lengueur	2	2
	Checksum	2	2
Total	8	8	
Données SNMP	Options	15 < correction < 21	15 < correction < 21
	entêtes	1	1
	Version	1	1
	Communauté (public)	6	6
	id requête	4	4
	Status erreur	1	1
	index erreur	1	1
	Variable Binding	16 x N	P x N (15 <P< 19)
	Total	14 + (16x N) +Correction	14 + (Px N) +Correction
	Total	56 + (16 x N) + Correction	56 + (P x N) + Correction

Tableau 7 : Détail d'un échange SNMP

La taille d'une requête ou d'une réponse SNMP se compose d'une partie commune de taille fixe (56 octets) correspondant aux entêtes des couches protocolaires et des champs SNMP, et d'une

partie de taille variable dépendant essentiellement du nombre des métriques collectés (N). On peut alors approcher la taille globale d'une requête/réponse SNMP par la fonction suivante:

$$\begin{aligned}
 \text{Taille}_{\text{Requête/Réponse}} &= \text{Taille}_{\text{requête}} + \text{Taille}_{\text{réponse}} \\
 &= (56 + I_{\text{Correction}} + 16 \times N) + (56 + I_{\text{Correction}} + P \times N) \\
 &\quad I_{\text{Correction}} : \text{indice de correction [entre 15 et 21 Octets]} \\
 &\quad P : \text{taille occupée par une métrique [entre 17 et 19 Octets]} \\
 \text{Taille}_{\text{Requête/Réponse}} &= 112 + (p+16) \times N + I'_{\text{Correction}} \\
 &\quad \left. \begin{array}{l} p \in \{17,18,19\} \\ I'_{\text{Correction}} \in [30,42] \end{array} \right\}
 \end{aligned}$$

Figure 61 : Calcul approximatif de la taille d'un échange SNMP

Cette formule définit un gabarit pour le volume de trafic généré par le standard SNMP. Ce gabarit est identifié par les deux fonctions suivantes:

$$\left\{ \begin{array}{l} \text{Volume}_{\text{SNMP min}} (\text{Octets}) = 142 + (33 \times N) \dots\dots | I'_{\text{Correction}} = 30 \dots P = 17 \\ \text{Volume}_{\text{SNMP max}} (\text{Octets}) = 154 + (35 \times N) \dots\dots | I'_{\text{Correction}} = 42 \dots P = 19 \end{array} \right.$$

Équation 6 : Approximation du volume de trafic généré par SNMP

Sous format graphique, cela donne les courbes présentées dans la Figure 62.

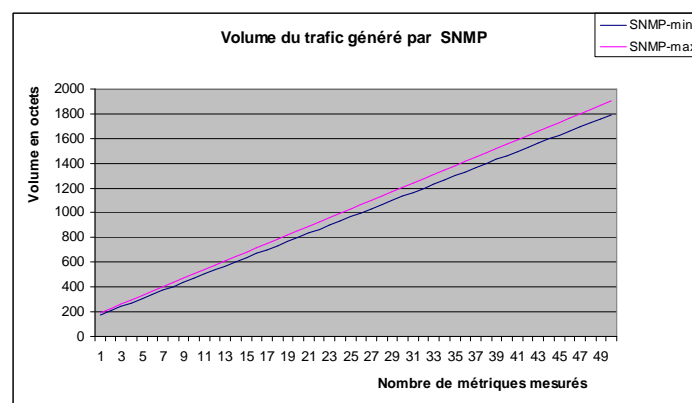


Figure 62 : Courbes des volumes de trafic générés par SNMP

Pour dix métriques mesurées, on obtient un volume de trafic compris entre 472 et 492 octets. Expérimentalement, ce volume vaut 486 octets.

2.3.3. Conclusion

De part leur mode de fonctionnement, les outils de supervision « online » génèrent un volume supplémentaire de trafic pendant la supervision. Le volume de trafic et donc le niveau d'intrusivité introduit par ces outils dépendent principalement du nombre des métriques mesurées et de la fréquence de la supervision.

L'analyse des échanges réseaux pour les deux frameworks TPTP et SNMP a permis de mesurer le trafic réseau généré par ces deux outils. La Figure 63 donne la représentation graphique des volumes de trafic générés par les outils étudiés avec une période de supervision d'une seconde.

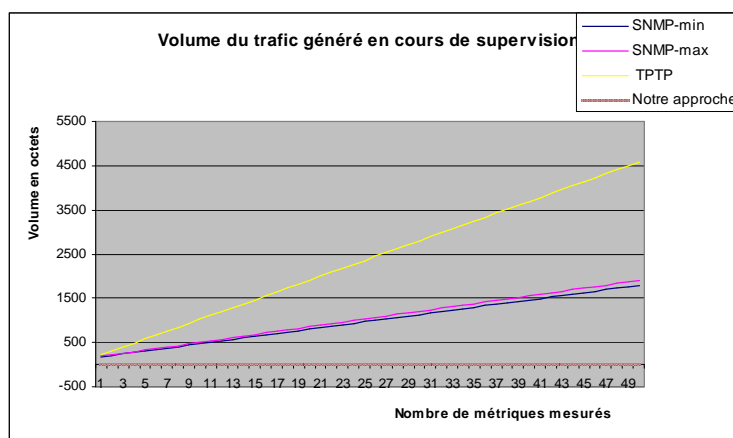


Figure 63 : Volume du trafic généré par TPTP et SNMP

Ces courbes montrent que l'outil TPTP génère plus de trafic que SNMP (soit un ratio de 2,5). Ceci est dû à l'utilisation du formalisme XML réputé d'être un langage verbeux.

2.3.4. Effet de l'intrusivité sur la bande passante

Pour consolider les résultats obtenus sur l'intrusivité des frameworks TPTP et SNMP nous avons mesuré la diminution de la bande passante (BW) disponible en présence de la supervision. Pour cela, nous avons mesuré à l'aide de l'outil *Iperf*⁷ la BW dans les cas suivants :

- ❖ *Pas de supervision,*
- ❖ *Supervision avec SNMP,*
- ❖ *Supervision avec TPTP,*
- ❖ *Supervision avec notre approche*

⁷ <http://dast.nlanr.net/Projects/Iperf/>

La plate-forme mise en œuvre pour réaliser cette expérimentation est décrite dans (Figure 64).

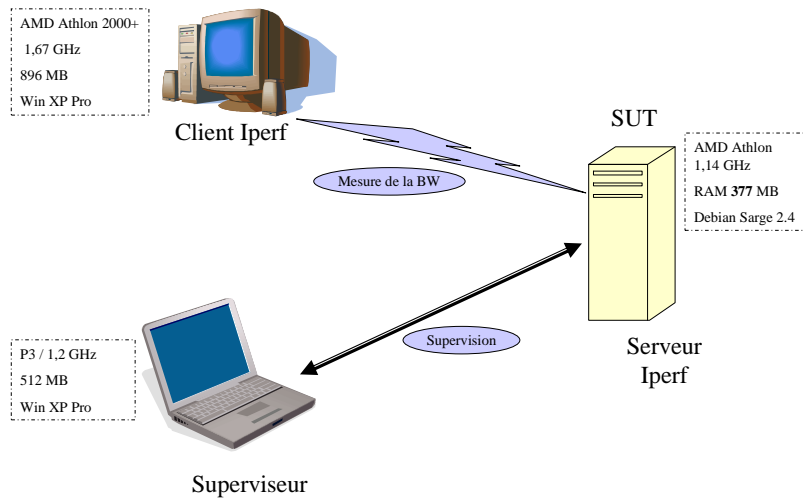


Figure 64 : Plate-forme de mesure de la bande passante

L'outil *Iperf* fonctionne selon un mode client/serveur. Sur la machine SUT, *Iperf* est exécuté en mode serveur avec l'option -s ou « --server » (Cf. Figure 65).

```
machine_sut $$ iperf --server
```

Figure 65 : Exécution de l'outil Iperf en mode serveur

Côté client, *Iperf* mesure le volume de trafic envoyé pendant un intervalle de temps. Cette expérimentation a été réalisée ~ 140 fois afin d'obtenir des mesures fiables. L'intervalle de temps est de 10 secondes. Les résultats obtenus sont présentés sous forme de graphe dans la Figure 66 et sous forme de tableau dans [Tableau 8].

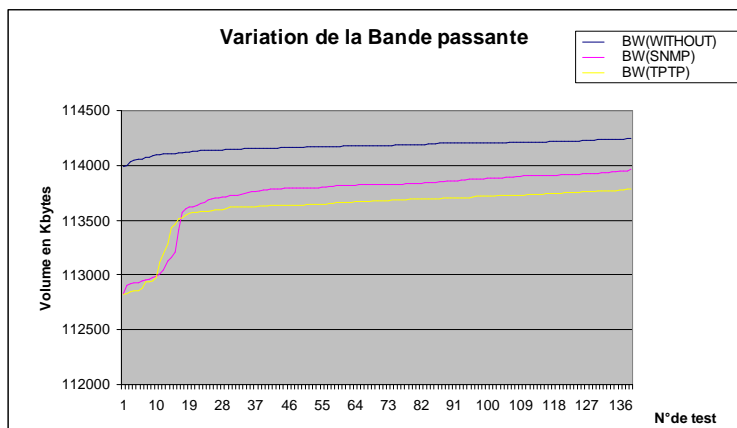


Figure 66 : Effet de la supervision sur la bande passante

	Sans Supervision	SNMP	TPTP
Volume du trafic (Octets)	114216	113912	113744
	114224	113912	113744
	114224	113912	113744
	114224	113912	113744
	114224	113920	113744
	114224	113920	113752
	114224	113920	113752
	114224	113920	113752
	114224	113920	113752
	114224	113920	113752
	114232	113928	113760
	114232	113928	113760
Moyenne (Octets)	114192	113849	113696
Moyenne (Mbit/s)	93,54	93,26	93,14
Dégradation (%)	X	0,30 %	0,43 %

Tableau 8 : Echantillon de mesures effectuées avec Iperf

Ces résultats consolident ce que nous avons constaté précédemment sur le fait que le protocole SNMP est moins verbeux que celui utilisé par TPTP. Bien qu'elles soient faibles, les diminutions de la bande passante engendrées par l'activation de la supervision (0,3 % pour SNMP et 0,43% pour TPTP) peuvent avoir un impact significatif dans un contexte de production où des dizaines, voire des centaines d'équipements sont supervisés en parallèles.

3. Conclusion générale

La mise en oeuvre expérimentale de notre approche de supervision « *offline* » a permis de la comparer à d'autres approches « *online* », à savoir SNMP et TPTP.

La comparaison de l'intrusivité au niveau des ressources systèmes a pu être réalisée en comparant « *l'overhead* » engendré par chaque solution. Cet « *overhead* » correspond à l'augmentation du temps d'exécution d'un script PHP utilisé comme application étalon. Les expérimentations ont prouvé une faible intrusivité de notre approche de supervision « *offline* » comparée aux approches « *online* » incarnées par le standard SNMP et le framework TPTP. De plus, notre approche, utilise d'une façon optimisée les ressources systèmes sur la machine superviseur au cours de la supervision. Ceci est dû à un découplage entre la machine superviseur et le SUT qui limite le rôle de cette première au début et à la fin de la supervision.

Pour ce qui est de l'utilisation des ressources réseaux, l'approche « *offline* » que nous proposons ne génère pas de trafic supplémentaire pendant la supervision. Les informations sont enregistrées localement puis traitées en fin de supervision ce qui permet une meilleure optimisation de la bande passante disponible.

G. Cas d'utilisation de la plate-forme de supervision

Ce chapitre a pour objectif de présenter des exemples illustrant l'utilisation et l'utilisabilité de la plate-forme de supervision dans des contextes différents. Le premier exemple concerne le test en charge où l'activité de supervision est très importante car elle permet d'observer l'utilisation des ressources systèmes et aide à l'analyse et l'interprétation des mesures fournies par les outils de test. Le deuxième exemple est un cas avancé de l'utilisabilité de la plate-forme qui se rapporte au domaine de l'observation active. L'idée consiste à coupler la sonde avec un composant d'alerte qui envoie une alarme lorsqu'un événement d'intérêt survient. En effet, notre approche « offline » ne s'oppose pas à la détection en temps-réel d'anomalies de fonctionnement de la QoS, et doit permettre la mise en œuvre de la réaction lorsqu'un problème est détecté.

1. Premier exemple : Test de montée en charge

1.1. Description de la plate-forme de test

L'exemple consiste à réaliser le test de montée en charge d'une application web. L'outil de test est OpenSTA et le but escompté est de pouvoir corréler l'ensemble des informations collectées sur le système sous test où elles sont générées automatiquement par l'outil de test en l'occurrence OpenSTA.

La plate-forme de test se compose de trois machines connectées entre elle via un réseau à haut débit. La configuration et les caractéristiques de ces machines sont décrites dans (Figure 67).

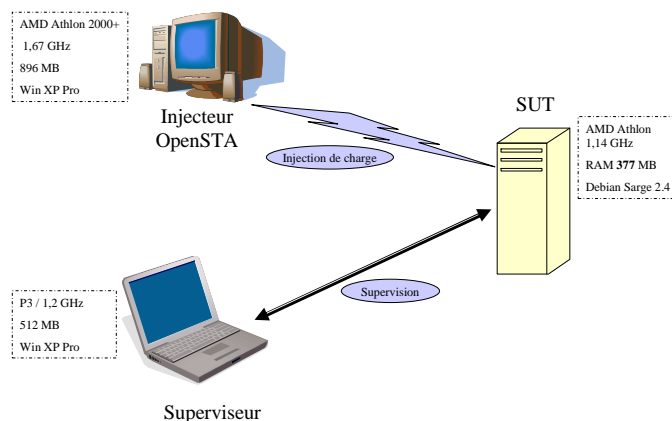


Figure 67 : Plateforme d'expérimentation

Sur la machine qui constitue notre SUT, nous déployons le serveur Tomcat (5.5.20) avec l'application **XPetStore** qui est une dérivée, à base de servlets, de l'application **Java PetStore** de Sun. Il s'agit d'un site web marchand où l'on peut choisir des animaux domestiques, les rajouter dans un panier, puis payer électroniquement.

Sur une deuxième machine, on installe l'outil **OpenSTA** qui va permettre de simuler un ou plusieurs utilisateurs accédant à l'application **XPetStore**. Cet outil sert aussi à mesurer le temps d'exécution de chaque action entreprise par l'utilisateur.

Sur la dernière machine (superviseur) est installée notre plate-forme de supervision avec les sondes *Vmstat* et *Ifstat* pour mesurer respectivement l'utilisation des ressources systèmes et réseaux.

1.2. Déroulement du test

Après la définition des scénarios de test dans **OpenSTA**, il faut spécifier le modèle de charge qui décrit comment les utilisateurs virtuels (ou VU pour Virtual User) accèdent à l'application **XPetStore** (simultanément, séquentiellement suivant une rampe). Chaque VU exécute le scénario suivant :

- ❖ *Création d'un compte et saisie des informations personnelles*
- ❖ *Connexion (authentification avec login / password)*
- ❖ *Achat d'un ou plusieurs articles*
- ❖ *Déconnexion*

OpenSTA mesure, pour chaque VU, le temps d'exécution de chaque phase du scénario. L'évolution des VU est mise à jour dans le fichier « VUsersLog.txt ». Ces informations sont à corréliser avec celles fournies par les sondes *Vmstat* et *Ifstat*. Pour la sonde *Vmstat*, on s'intéresse particulièrement à l'utilisation de la mémoire vive (RAM) de la mémoire en mode « swap » et du processeur. Pour la sonde *Ifstat*, les deux informations fournies (débit in, débit out) sont suffisantes pour évaluer l'utilisation du réseau.

En résumé, l'objectif de ce test est de disposer d'une cartographie globale qui reflète la dynamique de l'utilisation des ressources systèmes et réseaux sur le SUT en fonction du nombre d'utilisateurs (VU) utilisant l'application **XPetStore**. Cette cartographie doit permettre de s'assurer, par exemple, que les ressources systèmes et réseaux sont suffisantes pour supporter le nombre d'utilisateurs fixé. Il doit faciliter également une prise de décision sur l'évolution des performances de l'application par une augmentation judicieuse des ressources.

1.3. Résultats et analyses

Ci-dessous quelques exemples de graphes où différentes informations hétérogènes sont corrélées pour une analyse plus facile.

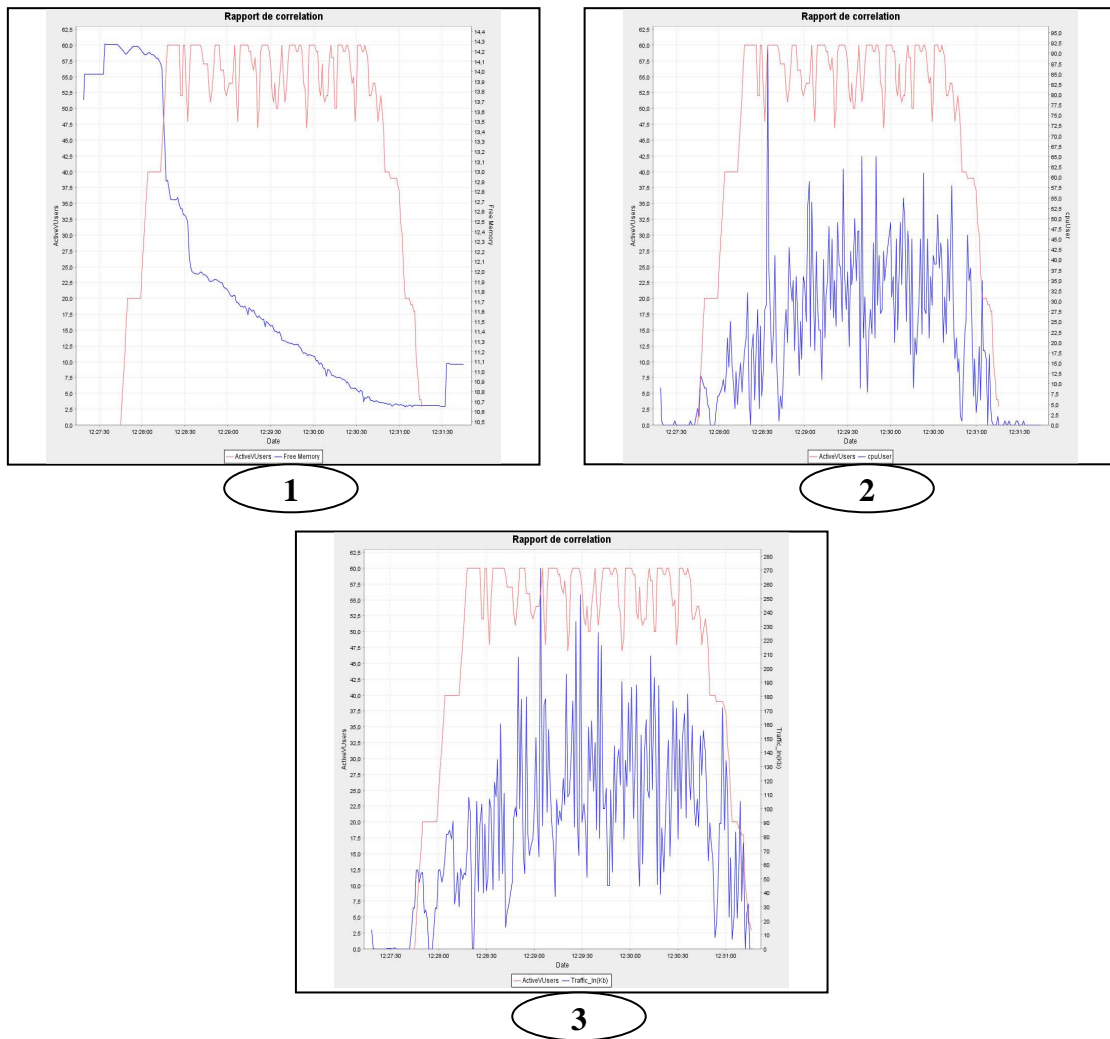


Figure 68 : Exemples de corrélation de

Le graphe 1 représente une corrélation du nombre des utilisateurs virtuels (VU = Virtuals Users) avec la mémoire physique disponible. Les courbes montrent une forte utilisation de la mémoire physique (~ 90 %) quand les soixante utilisateurs accèdent simultanément à l'application. Les graphes 2 et 3 représentent respectivement la corrélation du nombre (VU) avec l'utilisation du processeur et le trafic réseaux en entrée de la machine SUT. Ces différentes corrélations permettent de déduire qu'une éventuelle augmentation de la mémoire serait un choix judicieux pour augmenter les performances de l'application.

Avec les outils de supervision « online » tels que TPTP ou SNMP il serait difficile d'obtenir les mêmes résultats car certaines informations fournies par OpenSTA ne sont disponibles qu'à la fin du test ce qui limite leur utilisabilité. A noter que OpenSTA intègre son propre mécanisme de supervision basée sur SNMP et sur *Perfmon* pour les plates-formes Windows. Cependant, la corrélation des différentes informations n'est pas possible. La seule issue est d'exporter toutes ces informations en format CSV pour les exploiter avec un tableur Excel.

2. Deuxième exemple : plate-forme d'observation réactive

2.1. Description

Tout au long de ce manuscrit, nous avons décrit le rôle principal de la plate-forme qui est de pouvoir fédérer des sondes de supervision capables d'observer l'activité du système. Nous proposons à travers cet exemple d'étendre ce rôle en ajoutant un gestionnaire d'événements qui va assurer l'envoi des alertes vers diverses destinations. La description de cette nouvelle architecture est décrite dans la Figure 69.

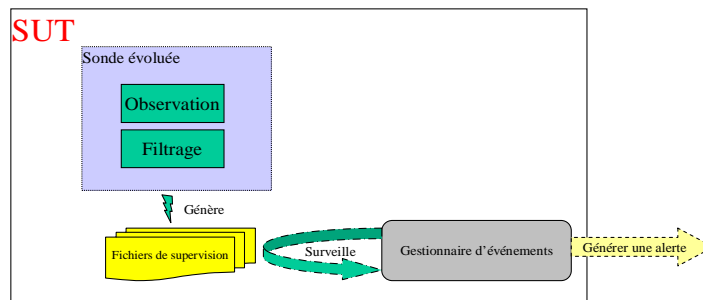


Figure 69 : Architecture d'une plateforme d'observation

Dans cette architecture, chaque sonde dispose d'un filtre qui permet d'enregistrer uniquement les événements qui représentent un intérêt (ex : mémoire physique supérieure à 95%). Le gestionnaire d'événements surveille, en continu, les fichiers de supervision et génère une alerte dès qu'un nouvel enregistrement est détecté.

2.2. Mise en œuvre

La mise en œuvre de cette solution requiert deux éléments :

- ❖ Une sonde dotée d'un filtre pour réduire les informations enregistrées. Le filtre peut être déclaré statiquement au moment du déploiement de la sonde.
- ❖ Un gestionnaire d'événement : une solution Open Source pour remplir cette tâche est l'outil SEC¹ (Simple Event Correlator). Cet outil, développé en perl et donc multi-plate-forme, utilise peu de ressources systèmes sur la machine où il est déployé [Vaarandi.'02]. Le détail de l'utilisation de SEC est présenté en annexe.

Cette plate-forme peut avoir plusieurs exemples d'applications notamment pour vérifier le respect des contrats de QoS (contrats portant sur l'utilisation des ressources du système). Grâce à la flexibilité de l'outil SEC, il est possible de personnaliser les actions à entreprendre en cas de violation de l'un des contrats.

¹ <http://sourceforge.net/projects/simple-evcorr/>

PARTIE 5
CONCLUSIONS &
PERSPECTIVES

H. Conclusion et perspectives

1. Conclusion

L'idée de mesurer les performances d'un système n'est pas nouvelle. Il existe des plates-formes et des standards qui ont été en partie présentés dans le chapitre « état de l'art » de ce mémoire. La difficulté réside cependant dans le fait que ces outils ne mesurent que des aspects bien particuliers des systèmes. Il faut souvent combiner plusieurs outils pour disposer d'une cartographie globale des performances d'un système. Cependant, cette solution se heurte au problème d'hétérogénéité des informations (ou mesures) fournies par ces outils en l'absence d'un consensus global de standardisation de ce type d'information. Nous sommes donc partis de ces constatations pour concevoir une plate-forme de supervision permettant de fédérer des outils de supervision. Cette plate-forme est simple d'utilisation et offre les services de supervision attendus par KEREVAL. Ces services permettent principalement de déployer des sondes, de rapatrier des résultats, d'analyser et enfin de corréliser des informations hétérogènes. La principale caractéristique de ce dernier service est le faible degré d'intrusivité qu'il engendre grâce à une approche « *offline* » de la supervision.

En effet, contrairement à une approche « *online* » où les informations collectées sont envoyées au fur à mesure de leur collecte, notre approche prévoit d'enregistrer ces informations dans des fichiers locaux qui seront rapatriés en fin de supervision pour être analysés. Certes, l'approche « *online* » à l'avantage de rendre disponible au contrôleur de supervision les informations en temps réel, mais cela se fait au détriment des ressources réseaux ce qui peut devenir vite un goulot d'étranglement pour certains types d'application (ex : application temps réel). Nous considérons que la réaction en temps-réel en cas de détection d'un non-respect de la QoS attendue peut se faire aussi dans une approche « *offline* » : les alertes sont soit traitées localement par une machine, soit lorsque cela est nécessaire envoyées au contrôleur pour une corrélation et une réaction globale. Dans notre approche, l'aspect « *offline* » ne pénalise donc pas une supervision réactive temps-réel, mais suppose de pouvoir établir a priori – sous forme de contrats de QoS par exemple – les seuils d'alerte à propager vers le contrôleur.

Outre l'intrusivité, une autre préoccupation lors de la conception de cette plate-forme est la possibilité de prendre en charge les différents formats des fichiers générés, soit par les sondes de la plate-forme, soit par des outils externes (ex : outils de test). Pour cela, nous avons intégré un module générique d'analyse des fichiers. Ce module est une évolution du module GLA (Generic Log Adapter) de la plate-forme TPTP. La problématique de synchronisation fut aussi un souci

dans notre thèse; la plate-forme est capable de corriger les décalages temporels qui peuvent subsister entre des fichiers générés sur des machines désynchronisées, sous réserve que la dérive d'horloge puisse être négligée dans la durée d'une supervision. La dérive moyenne étant de 10^{-6} , elle peut en général être négligée, sachant que l'échelle de temps qui nous intéresse est de l'ordre de la seconde.

La dernière partie du manuscrit, nous l'avons consacrée à la validation expérimentale des performances de la plate-forme et plus exactement à la comparaison de son degré d'intrusivité par rapport à d'autres techniques de supervision telles que TPTP et SNMP. Les résultats obtenus sont très encourageants et nous espérons améliorer davantage cet aspect en explorant d'autres pistes d'implémentation des sondes (ex : des sondes développées en langage C ou en Perl).

2. Perspectives

Ces travaux de thèse définissent une plate-forme pour fédérer des outils hétérogènes dans le but de mesurer les performances d'un système. Cette plate-forme peut prendre en charge et corréler des informations générées par des outils externes à la plate-forme ce qui constitue une condition sine qua non pour avoir une visibilité globale des performances du système. La plate-forme propose également des tâches pour déployer des sondes et rapatrier des fichiers de résultats.

De part sa conception et son implémentation (sous formes de tâches Ant), la plate-forme peut être intégrée facilement dans des applications où l'activité de supervision est très présente. Nous avons fait cet exercice avec le gestionnaire de test Salome TMF¹ et les conclusions sont prometteuses.

Nos travaux constituent une brique de base pour la construction d'un système plus complet de supervision « offline » avec les perspectives suivantes :

- ❖ *Offrir une solution adaptée pour vérifier les contrats de QoS, en proposant des mécanismes pour construire des sondes en fonction des contrats.*
- ❖ *Vérifier le bon fonctionnement de certaines contraintes comme « la répartition des charges » en observant l'équilibrage des sollicitations des ressources systèmes.*

¹ <https://wiki.objectweb.org/salome-tmf/>

PARTIE 6
ANNEXE

ANNEXE A. Network Time Protocol

PRESENTATION

Le protocole NTP (Network Time Protocol) est utilisé pour assurer la synchronisation des équipements réseau en permettant le transport des informations de temps entre ces équipements. Il est construit sur le protocole Internet (IP) et le protocole de datagramme d'utilisateur (UDP, User Datagram Protocol) qui fournit un mécanisme de transport sans connexion.

L'une des caractéristiques principales d'un réseau NTP est sa structure pyramidale. Un certain nombre de sources de référence, dites primaires, synchronisées par la radio ou un réseau filaire sur les standards nationaux, sont connectées à des ressources largement accessibles, comme des passerelles de backbone et des serveurs de temps primaires.

FONCTIONNEMENT

L'architecture de NTP est constituée d'un système de Strates (Figure 70)

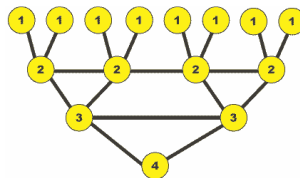


Figure 70 : Architecture NTP

Les serveurs primaires (strate 1), qui sont considérés comme des serveurs d'étalon, se synchronisent sur des horloges atomiques de référence. Les serveurs secondaires (strate 2) se synchronisent en mode client/serveur avec les serveurs primaires et en mode symétrique avec les autres serveurs secondaires. D'une manière générale, les serveurs d'une strate se synchronisent sur plusieurs serveurs de la strate supérieure en mode client/serveur, et avec d'autres serveurs de la même strate en mode symétrique.

Il existe alors trois modes de communication:

- Le mode client/serveur : Le client se synchronise sur l'heure du serveur.
- Le mode symétrique : La priorité est donnée au poste qui a la plus courte distance de synchronisation.
- Le mode multicast : Permet au client d'obtenir une réponse de plusieurs serveurs.

La précision du protocole NTP dépend des délais de transmission dans le réseau (i.e. de la charge du réseau) [Mills.'89]. Elle varie de quelques millisecondes, voire moins, dans un réseau LAN¹ jusqu'à quelques dizaines de millisecondes dans un réseau WAN².

Le protocole NTP est décrit dans la RFC 1119³, et ses applications sont diverses et en particulier dans les domaines nécessitant des précisions importantes sur l'horodatage (par exemple la métrologie réseau).

✚ ALGORITHME DE SYNCHRONISATION

Pour utiliser le format de message NTP afin de synchroniser les horloges d'ordinateurs, un programme **client** définit d'abord la zone Mode dans le format de message, pour indiquer qu'une demande est effectuée (mode client).

0		8		16		24	
LI	VN	Mode	Strate		Consultation		
Délai de racine (32)							
Dispersion de racine (32)							
Identifiant de référence (32)							
Horodatage de référence (64)							
Horodatage originel (64)							
Horodatage de réception (64)							
Horodatage d'émission (64)							
Authentifiant (facultatif) (96)							

Figure 71 : Trame NTP

Eventuellement, le client place l'heure courante de la machine cliente dans la zone d'horodatage. La zone « horodatage d'émission » pourra servir, lors du retour du message depuis le serveur NTP, à calculer le délai aller-retour de l'échange.

Après avoir défini ces zones, le client envoie le message au programme serveur NTP, en utilisant l'interface sockets et le port réservé à l'utilisation NTP, port numéro 123.

Le programme **serveur** NTP reçoit le message sur le port 123 puis règle la zone « horodatage de réception » d'après l'heure de réception de la requête.

Ensuite, la valeur du champ « horodatage d'émission » par le client est copiée dans la zone « horodatage original » d'où elle sera renvoyée au client et utilisée pour calculer le délai d'aller-

¹ Local Area Network

² Wide Area Network

³ <http://www.faqs.org/rfcs/rfc1119.html>

retour. Certaines zones de contrôle dans la première partie du message sont définies de manière à identifier le mode de fonctionnement du serveur.

Enfin, le serveur place l'heure courante dans le tampon horodateur «*horodatage d'émission*» et renvoie le message au client.

Dès réception, le **client** vérifie la validité du message, calcule le délai d'aller-retour et la différence entre les heures de l'horloge serveur et de l'horloge client (cette différence est appelée décalage d'horloge locale). Le client applique alors sa politique de mise à l'heure.

ANNEXE B. Oids des ressources systèmes

LOAD (CHARGE PROCESSEUR)

- 1 minute Load: .1.3.6.1.4.1.2021.10.1.3.1
- 5 minutes Load: .1.3.6.1.4.1.2021.10.1.3.2
- 15 minutes Load: .1.3.6.1.4.1.2021.10.1.3.3

CPU

- Percentage of user CPU time: .1.3.6.1.4.1.2021.11.9.0
- Rawuser cpu time: .1.3.6.1.4.1.2021.11.50.0
- Percentages of system CPU time: .1.3.6.1.4.1.2021.11.10.0
- Raw system cpu time: .1.3.6.1.4.1.2021.11.52.0
- Percentages of idle CPU time: .1.3.6.1.4.1.2021.11.11.0
- Raw idle cpu time: .1.3.6.1.4.1.2021.11.53.0

MEMORY STATISTICS

- Total Swap Size: .1.3.6.1.4.1.2021.4.3.0
- Available Swap Space: .1.3.6.1.4.1.2021.4.4.0
- Total RAM in machine: .1.3.6.1.4.1.2021.4.5.0
- Total RAM used: .1.3.6.1.4.1.2021.4.6.0
- Total RAM Free: .1.3.6.1.4.1.2021.4.11.0
- Total RAM Shared: .1.3.6.1.4.1.2021.4.13.0
- Total RAM Buffered: .1.3.6.1.4.1.2021.4.14.0
- Total Cached Memory: .1.3.6.1.4.1.2021.4.15.0

DISK STATISTICS

- Path of the device for the partition: .1.3.6.1.4.1.2021.9.1.3.1
- Total size of the disk/partition (kBytes): .1.3.6.1.4.1.2021.9.1.6.1
- Available space on the disk: .1.3.6.1.4.1.2021.9.1.7.1
- Used space on the disk: .1.3.6.1.4.1.2021.9.1.8.1

ANNEXE C. TPTP (Test & Performance Tools Platform)

🚩 COMMUNICATION CLIENT ↔ RAC

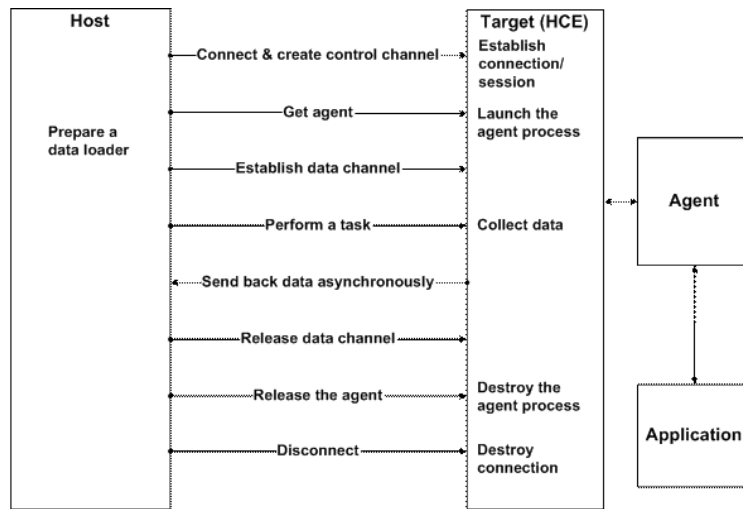


Figure 72 : Diagramme de communication entre le client et le RAC

Après lancement de l'agent contrôleur, ce dernier crée deux processus qui se mettent en écoute sur les ports « 10002 » et « 10005 » en attente de demande de connexion émanant du client. Le port « 10002 » est utilisé pour l'administration de l'agent contrôleur et le port « 10005 » connu sous le nom de « *filePort* » est celui utilisé pour le transfert de fichiers entre agent contrôleur et le client. Il est possible d'utiliser le port « 10003 » qui offre une connexion sécurisée entre le client et l'agent contrôleur.

Après établissement de connexion entre le client et l'agent contrôleur, le client sélectionne un agent qu'il souhaite déployer sur la machine cible. L'agent contrôleur lance un processus de collecte pour cet agent et établit un canal de donnée avec le client pour l'envoi des données de supervision.

Les informations de supervisions sont envoyées d'une manière asynchrone.

L'arrêt d'une supervision s'effectue en trois étapes:

- i. Libération du canal de données
- ii. Arrêt du processus agent
- iii. Libération du canal de contrôle. L'agent contrôleur reste en écoute sur le port « 10002 » en attente de nouvelles demandes de connexion.

ANNEXE D. NAGIOS

CODES RETOURS D'UN PLUG-IN NAGIOS

Codes de retour	Etat du service (ou ressource)	Description
0	OK	le pourcentage d'utilisation de la ressource n'a pas atteint les seuils prédéfinis
1	Warning	le pourcentage d'utilisation de la ressource a atteint le seuil « warning »
2	Critical	le pourcentage d'utilisation de la ressource a atteint le seuil « critique »
3	Unknown	Comportement indéterminé

ANNEXE E. FRACTAL & DREAM

FRACTAL

Fractal [Bruneton, et al.'02] est un modèle de composants développé par FT R&D et l'INRIA. Contrairement à d'autres modèles comme les EJB1 ou CCM2 dont les composants sont plutôt de grain moyen et destinés aux applications de gestion tournée vers l'Internet, la granularité des composants Fractal est quelconque. Fractal est organisé comme un projet du consortium *ObjectWeb* pour la construction des middlewares open source.

Fractal est composé de deux modèles: un modèle abstrait (ensemble de spécifications) et son implémentation dans différents langages de programmation comme Java, C, ou SmallTalk. Le modèle abstrait est indépendant de tout langage de programmation. Il définit un modèle de composants récursifs dans lequel les composants peuvent être assemblés pour former des composants de plus gros grain. Ces derniers ne sont en rien différents des premiers et peuvent à leur tour être assemblés (Figure 73)

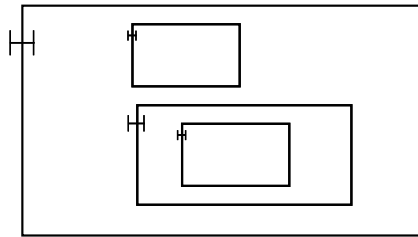


Figure 73 : Modèle de composant FRACTAL

Plusieurs niveaux de détails sont pris en compte dans Fractal et apportent des composants aux fonctionnalités de plus en plus riches. Au niveau le plus bas, un composant est une entité d'exécution (i.e. un objet Java). Ce niveau est explicité afin de faciliter l'intégration de l'existant. Au niveau suivant, les composants fournissent des fonctionnalités d'introspection, permettant une découverte de leurs interfaces. Le niveau suivant est un niveau de configuration qui permet de contrôler et de modifier le contenu (i.e. les sous-composants) d'un composant et leurs liaisons. Plusieurs types de contrôleurs sont disponibles : de contenu, de liaison, d'attributs, intercepteurs. Les niveaux suivants correspondent aux fonctionnalités d'instanciation et de typage des composants. Fractal fournit un langage de description

¹ Entreprise Java Beans

² Corba Component Model

d'architecture (ADL³) dont la syntaxe XML permet de décrire des assemblages de composants. Julia [Bruneton.'02] est l'implantation de Fractal en Java de FT R&D.

DREAM

DREAM est un framework pour les middlewares de communication, avec une préférence pour les MOM (middlewares orientés messages). L'objectif est de pouvoir construire, configurer et déployer ces middlewares, en s'appuyant sur le framework Fractal.

DREAM utilise la notion des *mixin*⁴ pour maximiser la factorisation de code tout en mettant en avant leur côté fonctionnel. Il réutilise l'API Socket, en l'emballant dans des services de flux de données, offerts aux applications. Ce framework est donc une surcouche à la pile réseau.

La bibliothèque DREAM contient des composants implantant diverses fonctions pouvant être assemblées pour former des intergiciels de communication. La bibliothèque DREAM se décompose en deux parties : un ensemble de composants permettant de manipuler des messages et un *canevas* pour la création de protocoles et de sessions à l'aide des composants de la bibliothèque.

Composants de la bibliothèque

Les composants de la bibliothèque traitent des messages qu'ils reçoivent par des interfaces particulières appelées *entrées* (Inputs) et qu'ils délivrent sur des interfaces appelées *sorties* (Output). Les messages sont des objets java qui encapsulent des sous-messages et des accesseurs (getter) et mutateurs (setter) pour modifier les propriétés des messages.

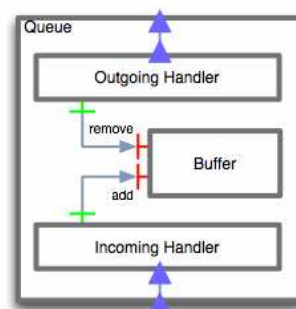


Figure 74 : Le composant File de message DREAM

D'autres exemples de composants qu'il est possible de trouver dans cette bibliothèque : les *files de messages* (queue) (Figure 74) servent à stocker les messages ; les *pompes* ont une activité qui

³ Architecture Description Language

⁴ Ce sont des classes abstraites qui implantent des fonctionnalités réutilisables.

consiste à récupérer un message sur leur entrée et à le délivrer sur leur sortie ; les *routeurs* routent les messages reçus sur leur entrée sur une ou plusieurs de leurs sorties.

Protocoles de liaisons

La bibliothèque DREAM définit des abstractions facilitant l'établissement de liaisons entre composants. DREAM permet de construire des protocoles qui permettent d'ouvrir des sessions au sein desquelles des messages peuvent être échangés.

ANNEXE F. Format CBE (Common Based Event)

Le CBE est le format commun d'échange des données au sein de la plate-forme TPTP. Ce formalise de données, qui a fait l'objet d'un draft pour OASIS WSDM⁵, tente d'apporter une solution au problème d'interopérabilité entre des applications hétérogènes.

Un événement CBE, assimilé dans notre cas à une information collectée, définit plusieurs champs notamment le champ « *CreationTime* » qui correspond à la date de collecte de l'information. Le schéma XML qui définit la structure d'un événement CBE est disponible ici⁶. Ci-joints des exemples de propriétés présentes dans la structure d'un événement CBE :

<i>Propriété</i>	<i>Description</i>
globalInstanceId	Identifiant global de l'événement. Ce dernier doit être unique
creationTime	Date de création de l'événement
Severity	Le niveau de sévérité de l'événement, compris entre 0 et 70 (ex 0-Unknown, 10-Information, 20-Harmless, 30-Warning, 40-Minor, 50-Critical and 60-Fatal)
Priority	La priorité (importance) de l'événement. Elle est comprise entre 0 et 100.
ReporterComponentId	L'identité du composant rapporteur de l'événement. Cette propriété est de type (<i>ComponentIdentification</i>) qui est un type de données complexe contenant des sous propriétés telle que <i>ProcessId</i> , <i>Location(@Ip)</i> , <i>ThreadId</i> , etc.
SourceComponentId	L'identité du composant affecté par l'événement
SituationType	Circonstance (ou cause) du déclenchement de l'événement (ex DEPENDENCY, REQUEST, CONFIGURE, CONNECT, CREATE et UNKNOWN)
Msg	Une description de l'événement, compréhensible à l'homme
extendedDataElements	Permet d'étendre la structure de CBE pour définir de nouvelles propriétés

⁵ Organisation for the Advancement of Structured Information Standards, Web Services Distributed Management

⁶ http://www.eclipse.org/tptp/platform/documents/resources/cbe101spec/commonbaseevent1_0_1.xsd

ANNEXE G. Les métriques collectées par Vmstat

Ci-dessous, vous trouverez une liste des métriques qui peuvent être mesurées à l'aide de la commande Vmstat. Cette liste n'est pas exhaustive et peut changer légèrement d'une distribution Linux à une autre.

🚦 *PROCESSUS*

- **r** : Nombre de processus dans la file d'exécution.
- **b** : Nombre de processus dormants (bloqués).
- **w** : Nombre de processus transférés dans le swap mais qui ne seraient pas exécutés.

🚦 *MEMOIRE*

- **swpd** : Quantité de mémoire virtuelle utilisée (ko)
- **free** : Quantité de mémoire libre (ko).
- **buff** : Quantité de mémoire utilisée comme tampons d'E/S (ko).

🚦 *SWAP*

- **si** : Quantité de mémoire paginée lue depuis un disque en ko/s.
- **so** : Quantité de mémoire paginée transférée sur disque en ko/s.

🚦 *INPUT / OUTPUT*

- **bi** : Blocs écrits par seconde sur des périphériques orientés bloc.
- **bo** : Blocs lus par seconde sur des périphériques orientés bloc.

🚦 *SYSTEM*

- **in** : Nombre d'interruptions par seconde, y compris l'horloge.
- **cs** : Nombre de changements de contextes (*context switches*) par seconde

🚦 *CPU*

- **us** (user time) : pourcentage de consommation en mode utilisateur
- **sy** (system time) : pourcentage de consommation en mode système
- **id** (idle time) : pourcentage de CPU non consommée
- **wa** : pourcentage de temps passé par la CPU à attendre le résultat de requêtes d'entrées/sorties.

ANNEXE H. Performance d'une analyse statique Vs une analyse basée sur des règles

L'objectif de cette comparaison est de mesurer les performances en terme de rapidité entre un « parseur » qui utilise une analyse statique (classe Java) et un « parseur » qui utilise des règles. Les deux techniques ont été appliquées sur des fichiers de tailles différentes (200 KB, 2 MB, 6 MB, 20 MB).

CONDITIONS EXPERIMENTALES

Hardware → les caractéristiques de la machine d'expérimentation sont les suivantes :

- Type du processeur : Intel® Pentium® M
- Vitesse du processeur : 1.59 GHz
- RAM : 1.5 MB

Software → les logiciels utilisés :

- OS : Windows XP Pro SP1
- Version du JRE : 1.4.2
- Version du « GLA » : 3.0.0 Bildt id: 20040831_1304

	200 KB (1538 records)	2 MB (15380 records)	6 MB (46096 records)	20 MB (153602 records)
Analyse statique (AS)	1 sec	6 sec	19 sec	74 sec
Analyse basée sur les règles (ABR)	3 sec	75 sec	462 sec	Non défini
rapport de performance (AS/ABR)	3 fois plus rapide	12 fois plus rapide	24 fois plus rapide	Non défini

Tableau 9 : Comparaison des performances des "parseurs" dans le module GLA

Le résultat complet de cette comparaison est disponible sur le site d'IBM à cette adresse¹.

¹ <http://www-128.ibm.com/developerworks/autonomic/library/ac-time/index.html>

ANNEXE I. Simple Event Correlator

SEC est un outil open source de corrélation d'évènements ayant recours à une approche de règles pour le traitement des évènements. Cette approche a été choisie en raison de sa facilité à représenter les connaissances et de la transparence de son processus de corrélation des évènements.

SEC reçoit ses évènements d'entrée de flux de fichiers. Il est ainsi possible d'utiliser SEC en tant que solution de surveillance des journaux d'évènements afin de l'intégrer avec n'importe quelle application capable de rédiger ses évènements de sortie sur un flux de fichiers (ce qui est vrai pour notre plate-forme).

SEC peut produire des évènements de sortie en exécutant des commandes shell définies par l'utilisateur, en rédigeant des messages dans les fichiers ou dans les canaux nommés, ou en appelant des sous-programmes. Les évènements de sortie peuvent également être envoyés sur le réseau vers une autre instance de SEC.

Les paramètres de configuration de SEC sont stockés dans des fichiers texte. Chaque fichier de configuration contient une ou plusieurs règles, et les ensembles de règles fournis par les fichiers sont appliqués virtuellement en parallèle. Ces règles peuvent contenir des expressions régulières ce qui rend plus flexible son utilisation. L'exemple ci-dessous donne un exemple simple du fichier de configuration de SEC.

```
## File sec_cfg #####  
  
type=Single  
ptype=TValue  
pattern=TRUE  
desc=violation contrat  
action= mail -s'violation contrats ' root
```

Figure 75 : Exemple d'un fichier de configuration SEC

SEC s'exécute de la façon suivante :

```
mhi@pcmhi:$ perl sec.pl -conf=sec_cfg -input=toto.txt
```

L'exemple ci-dessus correspond à l'envoi d'un mail à root à chaque fois qu'il y a une nouvelle écriture dans le fichier *toto.txt*

PARTIE 7
BIBLIOGRAPHIE

- [Al-Shaer.'96] - Ehab S. Al-Shaer, "High-Performance Event Filtering: Survey and Evaluation," 1996.
- [Amorim.'04] - Karine Macedo De Amorim, "Modélisation d'aspects qualité de service en UML: application aux composants logiciels," Rapport de thèse. 2004
- [Bruneton.'02] - Eric Bruneton, "Julia Tutorial," 2002.
- [Bruneton, et al.'02] - Eric Bruneton, Thierry Coupaye and Jean-Bernard Stephani, "The Fractal Composition Framework," 2002.
- [CCITT.'89] - CCITT, "General Aspects of Quality of Service and Network Performance in Digital Networks," 1989.
- [Chakraborty, et al.'03] - Debasish Chakraborty, Goutam Chakraborty and Norio Shiratori, "A dynamic multicast routing satisfying multiple QoS constraints," *International Journal of Network Management* vol. 13, 2003.
- [Collet and Rousseau.'05] - Philippe Collet and Roger Rousseau, "ConFract, un système pour contractualiser des composants logiciels hiérarchiques," in *Langages et Modèles à Objets, LMO'05*, 2005.
- [Curtis and David.'02] - Smith Curtis and Henry David, "High-performance Linux Cluster Monitoring Using Java," in *Proceedings of the 3rd Linux Cluster International Conference*, 2002.
- [Debeaupuis and Abela.] - Tristan Debeaupuis and Jérôme Abela, "Universal Format for Logger Messages," IETF Internet Draft.
- [Goutelle and Primet.'03] - Mathieu Goutelle and Pascale Primet, "Study of a non intrusive and accurate method for measuring the end-to-end useful bandwidth," INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE (INRIA) 2003.
- [Himdi.'04] - Marouane Himdi, "Development of Generic Probes for Functional and extra-Functional Diagnosis," in *International Symposium on Software Reliability Engineering (ISSRE), "Student paper"*. St-Malo, 2004.
- [ISO.'95] - ISO, QoS Framework, ISO/IEC/JTC1/SC21/WG1 N9680, 95.
- [ISO.'97] - ISO, "Information Technologie - Quality of Service - Framwork , ," Geneva, 1997.
- [Ivan, et al.'99] - Zoraja Ivan, Rackl Günther and Ludwig Thomas, "Towards Monitoring in Parallel and Distributed Environnement," in *International Conference on Software in Telecommunications and Computer Networks - SoftCom '99* 1999, pp. 133-141.
- [Jain.'91] - Ray Jain, The art of computer systems performance analysis, Techniques for experimental design, measurement, simulation, and modelling, John WILEY ed., 1991.
- [Jezequel.'89] - Jean-Marc Jezequel, "Building a global time en parallel machines," in *3rd International Workshop on Distributed Algorithms*, 1989, pp. 136--147.
- [Katchabaw, et al.'96] - Michael J. Katchabaw, Stephen L. Howard, Andrew D. Marshall and Michael A. Bauer, "Evaluating the costs of management: a distributed applications

- management testbed," in *Centre for Advanced Studies on Collaborative research*, 1996.
- [Kephart and Chess.'03] - Jeffrey O Kephart and David M Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, 2003.
- [Kleinrock.'75] - Leonard Kleinrock, *Queuing Systems - Theory* vol. 1: Wiley-Interscience, 1975.
- [Knop, et al.'02] - Mike Knop, Praveen Paritosh, Peter Dinda and Jennifer Schopf, "Windows Performance Monitoring and Data Reduction using WatchTower," in *Workshop on Self-Healing, Adaptive, and sel-Managed Systems (SHAMAN 2002)*, New York, 2002.
- [Leclercq, et al.'04] - Matthieu Leclercq, Vivien Quéma and Jean-Bernard Stefani, "DREAM: a component framework for the construction of resource-aware, reconfigurable MOMs," in *3rd workshop on Adaptive and reflective middleware*, 2004.
- [Mansouri-Samani.'95] - Masoud Mansouri-Samani, "Monitoring of Distributed Systems," Rapport de thèse. 1995
- [Marsan, et al.'95] - M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli and G. Franceschinis, *Modeling with Generalized Stochastic Petri Nets*: John Wiley & Sons, 1995.
- [Mills.'89] - D.L Mills, "Measured Performance of the Network Time Protocol in the Internet System," 1989.
- [Nicolas.'04] - LE SOMMER Nicolas, "Contractualisation des ressources pour les composants logiciels : une approche réflexive," Rapport de thèse. 2004
- [Owezarski.'03] - Philippe Owezarski, "Métrologie des réseaux de l'Internet: principales actions et impact sur les évolution technologiques," in *Colloque International "Mesures de l'Internet"*, Nice, FRANCE, 2003.
- [Pattinson.'01] - Colin Pattinson, "A Study Of the Behaviour of the Simple Network Management Protocol," in *12th International Workshop on Distributed Systems: Operations & Management (DSOM)*, Nancy (France), 2001.
- [Pras, et al.'04] - Aiko Pras, Thomas Drevers, Remco van de Meent and Dick Quartel, "Comparing the Performance of SNMP and Web Services-Based Management," in *IEEE electronic Transactions on Network and Service Management*, 2004.
- [Ramchurn, et al.'04] - Sarvapali D. Ramchurn, Benjamin Deitch, Mark K. Thompson, David C. De Roure, Nicholas R. Jennings and Michael Luck, "Minimising Intrusiveness in Pervasive Computing Environments using Multi-Agent Negotiation," in *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, 2004.
- [RFC.1157.] - RFC.1157, "Simple Network Management Protocol (SNMP)."
- [Subramanyan, et al.'00] - Rajesh Subramanyan, Jose Miguel-Alonso and Jose A.B. Fortes, "Design and evaluation of a SNMP-based monitoring system for heterogeneous, distributed computing," Purdue University School of Electrical and Computer Engineering 2000.

- [Vaarandi.'02] - Risto Vaarandi, "SEC - a LIGHTWEIGHT Event Correlation Tool," in *Workshop on IP Operations and Management*, 2002.
- [Vogel, et al.'95] - Andreas Vogel, Brigitte Kerhervé and Gregor von Bochmann, "Distributed multimedia applications and Quality of Service: - A survey " *IEEE Mutimedia Journal*, 1995.

Abstract

Measuring performances is an important task for understanding its dynamic behavior and then to anticipate its evolution. Such task requires monitoring tools which take into account the properties both of the application and of its deployment environment, such as distribution and heterogeneity. These tools should also have as low intrusiveness as possible to avoid the disruption of the monitored system.

Our contribution in this thesis is to design and implement a monitoring platform based on the dynamic deployment of probes which gather information about the use of system's resources. These probes have been chosen because of their low intrusiveness. To decrease the network exchanges, instead of transmitting data directly to the monitor each probe keeps data locally without any prior processing. The analysis treatments are postponed, and done *offline*. This monitoring strategy allows the optimization of bandwidth and system resources. To deal with heterogeneity, the monitoring platform includes a generic parser engine which facilitates the transformation of heterogeneous gathered data into a common format. This common format is based on XML and the correlation processing are performed at this level of genericity.

This thesis defines a platform which federates probes monitoring. The concept of this solution can be extended to assist the construction of systems needing to check the status of resources. This is the case of systems that monitor performance contracts.

Keywords

Distributed system, monitoring, probes, quality of service, performances, generic framework, intrusiveness and offline correlations.

Résumé

Mesurer les performances d'une application est devenue une tâche incontournable pour mieux comprendre sa dynamique et anticiper son évolution. Cette tâche nécessite des outils de supervision qui s'adaptent aux différentes spécificités de l'application et de son environnement de déploiement (distribution, hétérogénéité, etc.). Ces outils doivent également être les moins intrusifs possibles pour ne pas perturber le fonctionnement de l'application.

Notre contribution dans cette thèse est de concevoir et d'implémenter une plateforme de supervision basée sur le déploiement dynamique de sondes pour collecter des informations sur l'utilisation des ressources systèmes. Ces sondes sont choisies pour leurs faibles degrés d'intrusivité. Elles fonctionnent en mode « offline » en enregistrant localement et sans traitement préalable les informations collectées au lieu de les transmettre sur le réseau. Ce mode de fonctionnement est moins intrusif car il optimise l'utilisation des ressources systèmes et réseaux. La plateforme intègre également un moteur générique d'analyse qui assure la transformation des formats hétérogènes des informations collectées vers un format pivot basé sur XML ce qui facilite la phase de corrélation.

Ce travail de thèse, réalisé dans un cadre industriel, définit une plateforme pour fédérer des sondes de supervision. Le concept de cette plateforme peut être étendue pour faciliter la construction de systèmes nécessitant de vérifier l'état des ressources. C'est le cas par exemple des systèmes qui vérifient les contrats de performances.

Mots clés

Systèmes distribués, sondes de supervision, qualité de service, performances, plateforme générique, intrusivité, corrélation offline.