Design and Implemention of a Plugin Scheduler for DIET

March 11, 2005

Outline

1. Background on DIET
   - Computational Grid Computing
   - DIET Framework
   - Motivation for Plugin Scheduler

2. Plugin Scheduler
   - Design
   - Implementation
   - Current Status
   - (Near-)Future Work

Outline

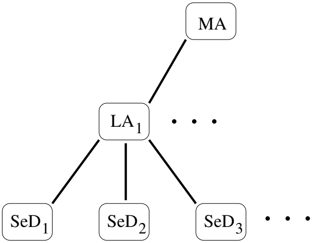
1. Background on DIET
   - Computational Grid Computing
   - DIET Framework
   - Motivation for Plugin Scheduler
2. Plugin Scheduler
   - Design
   - Implementation
   - Current Status
   - (Near-)Future Work

The Computational Grid and DIET

- Grid platforms
  - heterogeneous computational resources
  - irregular network topologies
  - dynamic resource performance
- DIET philosophy and design principles
  - server and broker agent model
  - hierarchical organization
  - flexible deployment options

DIET Overview

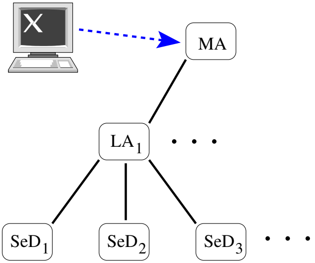
DIET hiearachy:



Basic progress of a DIET call:

DIET Overview

DIET hieararchy:

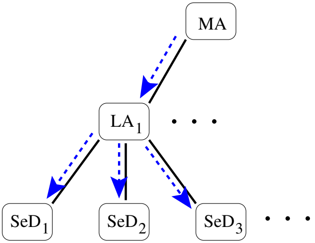


Basic progress of a DIET call:

- Client requests service from the Master Agent (MA)

DIET Overview

DIET hieararchy:

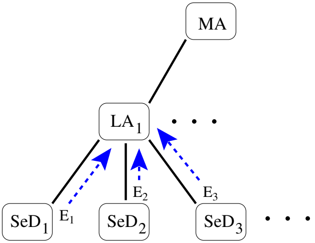


Basic progress of a DIET call:

- Client requests service from the Master Agent (MA)
- The MA interrogates the DIET hierarchy

DIET Overview

DIET hiearachy:

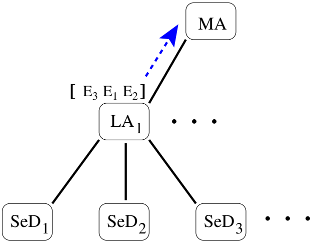


Basic progress of a DIET call:

- Client requests service from the Master Agent (MA)
- The MA interrogates the DIET hierarchy
- Each Server Daemon (SeD) responds with an execution time estimate

DIET Overview

DIET hieararchy:

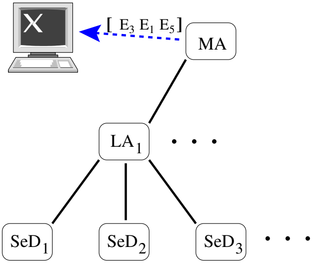


Basic progress of a DIET call:

- Client requests service from the Master Agent (MA)
- The MA interrogates the DIET hierarchy
- Each Server Daemon (SeD) responds with an execution time estimate
- Each Local Agent (LA) compiles and sorts the responses by execution time

DIET Overview

DIET hieararchy:

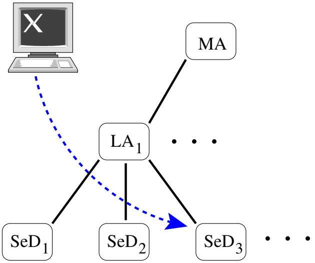


Basic progress of a DIET call:

- Client requests service from the Master Agent (MA)
- The MA interrogates the DIET hierarchy
- Each Server Daemon (SeD) responds with an execution time estimate
- Each Local Agent (LA) compiles and sorts the responses by execution time
- MA returns a list of servers to the client

DIET Overview

DIET hieararchy:



Basic progress of a DIET call:

- Client requests service from the Master Agent (MA)
- The MA interrogates the DIET hierarchy
- Each Server Daemon (SeD) responds with an execution time estimate
- Each Local Agent (LA) compiles and sorts the responses by execution time
- MA returns a list of servers to the client
- Client launches service directly on SeD

Some Implementation Details

Three primary components of the DIET system:

Some Implementation Details

Three primary components of the DIET system:

- Agents (MA and LA)
  - implemented in C++
  - scope: DIET internal
  - at runtime, assembled top-down

Some Implementation Details

Three primary components of the DIET system:

- Agents (MA and LA)
    - implemented in C++
    - scope: DIET internal
    - at runtime, assembled top-down
- Servers (SeD)
    - implemented in either C or C++
    - scope: application service developer API
    - hierarchy must exist

Some Implementation Details

Three primary components of the DIET system:

- Agents (MA and LA)
  - implemented in C++
  - scope: DIET internal
  - at runtime, assembled top-down
- Servers (SeD)
  - implemented in either C or C++
  - scope: application service developer API
  - hierarchy must exist
- Client
  - implemented in either C or C++
  - scope: application developer or user
  - uses services existing at execution time

Some Implementation Details

Three primary components of the DIET system:

- Agents (MA and LA)
  - implemented in C++
  - scope: DIET internal
  - at runtime, assembled top-down
- Servers (SeD)
  - implemented in either C or C++
  - scope: application service developer API
  - hierarchy must exist
- Client
  - implemented in either C or C++
  - scope: application developer or user
  - uses services existing at execution time

Communication infrastructure

- CORBA-based model
- omniORB implementation

Advantages and Limitations

Advantages:

- scalability: hierarchy enables parallel server interrogation and distributed scheduling of requests
- straighforward interface: just the name and the correct number of arguments are needed
- abstraction: distributed platform details are largely hidden

Advantages and Limitations

Advantages:

- scalability: hierarchy enables parallel server interrogation and distributed scheduling of requests
- straightforward interface: just the name and the correct number of arguments are needed
- abstraction: distributed platform details are largely hidden

Limitations:

- deployment of appropriate hierarchies for a given grid platform is non-obvious
- limited consideration of inter-task factors
- non-standard application-specific performance measures

Advantages and Limitations

Advantages:

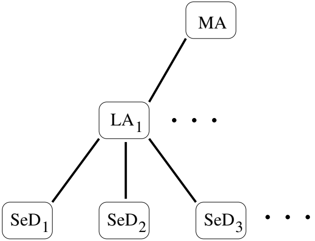
- scalability: hierarchy enables parallel server interrogation and distributed scheduling of requests
- straighforward interface: just the name and the correct number of arguments are needed
- abstraction: distributed platform details are largely hidden

Limitations:

- deployment of appropriate hierarchies for a given grid platform is non-obvious
- limited consideration of inter-task factors
- non-standard application-specific performance measures

Application-specific Performance Use Case

DIET hieararchy:

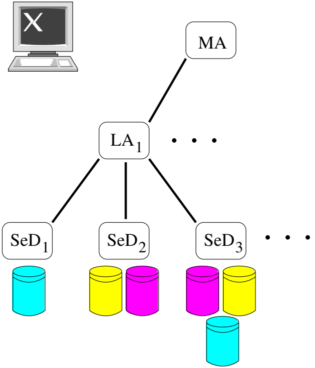


Motivation

- basic DIET deployment

Application-specific Performance Use Case

DIET hieararchy:

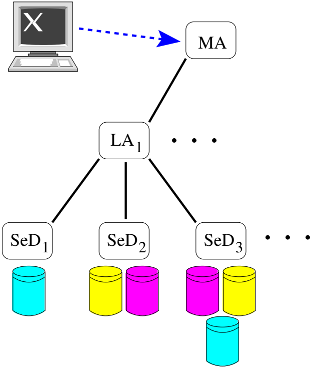


Motivation

- basic DIET deployment
- client application with data dependencies

Application-specific Performance Use Case

DIET hieararchy:

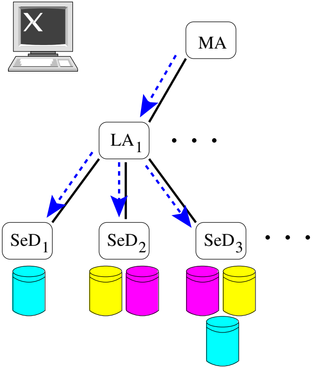


Motivation

- basic DIET deployment
- client application with data dependencies

Application-specific Performance Use Case

DIET hieararchy:

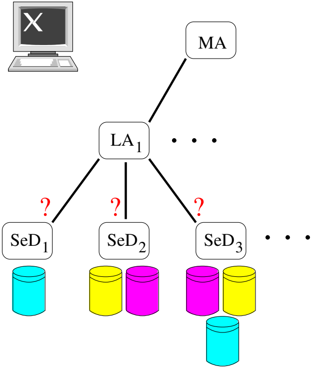


Motivation

- basic DIET deployment
- client application with data dependencies

Application-specific Performance Use Case

DIET hiearachy:

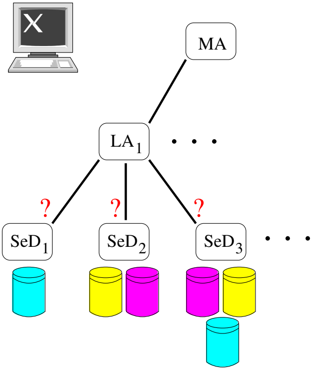


Motivation

- basic DIET deployment
- client application with data dependencies
- “performance” is not well-defined

Application-specific Performance Use Case

DIET hieararchy:



Motivation

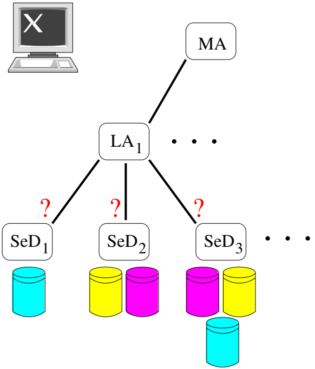
- basic DIET deployment
- client application with data dependencies
- "performance" is not well-defined

Possible meanings for performance

- existence of data
- avail. free memory
- specific architecture
- previous scheduling decisions
- application-specific measures
- composite requirements
- ...

Application-specific Performance Use Case

DIET hieararchy:



Motivation

- basic DIET deployment
- client application with data dependencies
- "performance" is not well-defined

Possible meanings for performance

- existence of data (GriPPS)
- avail. free memory (MUMPS?)
- specific architecture (TLSE)
- previous scheduling decisions
- application-specific measures
- composite requirements
- ...

Outline

1. Background on DIET
   - Computational Grid Computing
   - DIET Framework
   - Motivation for Plugin Scheduler
2. Plugin Scheduler
   - Design
   - Implementation
   - Current Status
   - (Near-)Future Work

Plugin Scheduling

Plugin scheduling facilities to enable

- application-specific definitions of appropriate performance metrics
- an extensible measurement system
- tunable comparison/aggregation routines for scheduling

Plugin Scheduling

Plugin scheduling facilities to enable

- application-specific definitions of appropriate performance metrics
- an extensible measurement system
- tunable comparison/aggregation routines for scheduling

Design changes

Component	Before	After
SeD	automatic performance estimate (FAST/NWS)	chosen/defined by application programmer

Plugin Scheduling

Plugin scheduling facilities to enable

- application-specific definitions of appropriate performance metrics
- an extensible measurement system
- tunable comparison/aggregation routines for scheduling

Design changes

Component	Before	After
SeD	automatic performance estimate (FAST/NWS)	chosen/defined by application programmer
Agents	exec. time sorting	"menu" of aggregation methods

Plugin Scheduling

Plugin scheduling facilities to enable

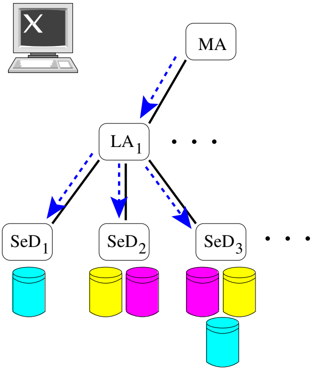
- application-specific definitions of appropriate performance metrics
- an extensible measurement system
- tunable comparison/aggregation routines for scheduling

Design changes

Component	Before	After
SeD	automatic performance estimate (FAST/NWS)	chosen/defined by application programmer
Agents	exec. time sorting	"menu" of aggregation methods
Client	CLIENT CODE UNCHANGED	

Plugin Scheduling Enhancements

DIET hieararchy:

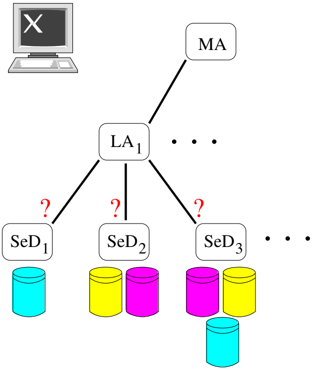


Example: Client request for comparison operation on blue database

- request arrives at SeD level

Plugin Scheduling Enhancements

DIET hieararchy:

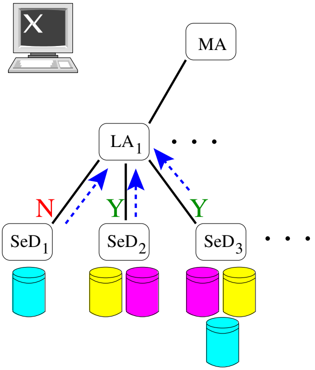


Example: Client request for comparison operation on blue database

- request arrives at SeD level

Plugin Scheduling Enhancements

DIET hieararchy:

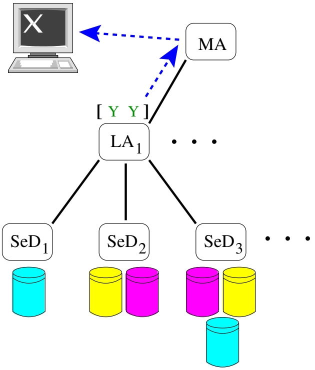


Example: Client request for comparison operation on blue database

- request arrives at SeD level

Plugin Scheduling Enhancements

DIET hieararchy:

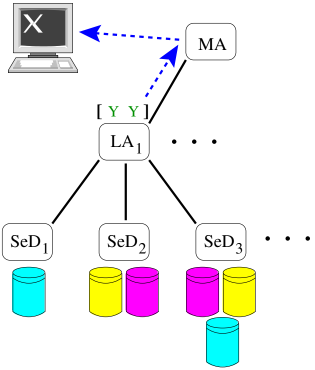


Example: Client request for comparison operation on blue database

- request arrives at SeD level
- only positive responses need to be propagated through the hierarchy

Plugin Scheduling Enhancements

DIET hieararchy:



Example: Client request for comparison operation on blue database

- request arrives at SeD level
- only positive responses need to be propagated through the hierarchy
- simple example: client gets random choice of two feasible servers
- more realistic: other factors used to decide
  - processor speed, memory
  - database contention
  - future requests

Implementation Mechanisms

What mechanisms are needed to implement this framework?

- **SeD-level** (response to client request)
  - interrogate the system performance
  - store selected performance metrics
- **Agent-level** (aggregation of server responses)
  - collect server responses and extract stored performance estimates
  - order responses from children, based on provided metrics
  - forward ordered responses to next higher level

SeD-level Interface

Estimation Vector

- Dynamic array of *estimation values*:
    - tag (byte) + value (float)
    - `estVector_t new_estVector()`
    - `int estVector_addEstimation(estVector_t, diet_est_tag_t, double)`

SeD-level Interface

Estimation Vector

- Dynamic array of *estimation values*:
    - tag (byte) + value (float)
    - `estVector_t new_estVector()`
    - `int estVector_addEstimation(estVector_t, diet_est_tag_t, double)`
- Tags and access functions for existing performance metrics
    - FAST/NWS (e.g, `int diet_estimate_fast(estVector_t, const diet_profile_t*)`)
    - SeD execution timestamp (to approximate Round-robin scheduling)

SeD-level Interface

Estimation Vector

- Dynamic array of *estimation values*:
  - tag (byte) + value (float)
  - `estVector_t new_estVector()`
  - `int estVector_addEstimation(estVector_t, diet_est_tag_t, double)`
- Tags and access functions for existing performance metrics
  - FAST/NWS (e.g, `int diet_estimate_fast(estVector_t, const diet_profile_t*)`)
  - SeD execution timestamp (to approximate Round-robin scheduling)
- User-defined tags

SeD-level Interface

Estimation Vector

- Dynamic array of *estimation values*:
  - tag (byte) + value (float)
  - `estVector_t new_estVector()`
  - `int estVector_addEstimation(estVector_t, diet_est_tag_t, double)`
- Tags and access functions for existing performance metrics
  - FAST/NWS (e.g, `int diet_estimate_fast(estVector_t, const diet_profile_t*)`)
  - SeD execution timestamp (to approximate Round-robin scheduling)
- User-defined tags
- Equivalent CORBA object and marshalling function

Agent-level Interface

New Profile Parameters

- New dynamic array of prioritized *optimization directives*:
  - *tag*: basis for comparison
  - *semantics*: maximize, minimize, etc.
- At service registration time, directives are fixed
- At runtime, directives used to order server responses

Modules' Status

Estimation Vector

- Fully functional API for storage of raw values
- Basic library of standard estimators
- Interface for user-defined metrics to be redesigned

DIET Profile Enhancements

- Existing scheduling strategy (i.e., preference for FAST) re-implemented using estimation vector
- User-defined metrics currently ignored
- Providing access to DIET agent hiearchy not previously supported

Work in Progress

Near-term Milestones

- Profile parameter extension to support priority optimization
- New performance estimator routines
    - alternative performance measurement systems (e.g., ganglia)
    - emerging DIET functionality (e.g., SeD-level queues)
- Initial plugin scheduler: DIET release 2.0

Open Issues

- Enforcement of optimization strategy over entire hiearchy
- Evaluate need for more expressive aggregation methods
- Incorporate runtime scheduling preferences