Performance Characterization of Metacomputing Systems

or

How we are approaching the performance study of a metasystem

Emilio Hernández and Mariela Curiel
Universidad Simón Bolívar
Caracas, Venezuela

Agenda

- Performance modeling issues in metasystems
- Application performance
- Metasystem performance
- Case study: SUMA (Scientific Ubiquitous Metacomputing Architecture)

Performance modeling issues in metasystems

- Applications
  - Different sources of overhead
  - Standardization of metrics on heterogeneous platforms
  - Need of both deterministic and probabilistic performance models

Metasystem performance issues

- Metasystem itself
  - Define what is metasystem performance
  - Need of services for estimating metasystem performance factors
  - Impact of parallel execution models on networks designed for client-server execution models

Application Performance

Application performance components

- Metasystem overhead
  - Queuing, file transfers, communication setup, compilation of portable code, results transfer, etc.
- Overhead produced by external load
  - Node sharing, network sharing. etc.
- Application performance
  - Communication, computation, I/O

Application performance

- It is hard to model/predict application performance if
  - execution platform power is not known
  - actual execution platform changes every time we run an application
  - metasystem overhead is not known
  - we do not know how external load affects the execution of applications

Application performance

- execution platform power is not known
  - execution platform performance should be modeled, this is easier if platforms are predefined
  - there should be services to know the performance parameters of the execution platforms
  - the users should be able to specify (minimum) performance requirements: they could predict (an upper bound of) execution time

Application performance

- actual execution platform changes every time we run our application
  - we can provide services for selecting previous used platform
  - we could define "similar" platforms in terms of performance => the users could define performance requirements instead of selecting previous used platforms

Application performance

- Metasystem overhead is not known
  - we must model "pure" metasystem overhead
    - queuing time
    - program and data transfer time
    - other "administrative" time, e.g. authentication
  - we should provide services to obtain information on metasystem overhead

Application performance

- external load affects the execution of applications
  - isolate platforms from external load
  - characterize external loads in order to model impact on application performance
  - we can define "isolation" levels for platforms
    - I1: isolated platforms, no external load
    - I2: platforms in which we can model external load
    - I3: platforms in which we can not model external load

An approach for performance analysis/modeling

- A metasystem is a complex system.
  - Divide and conquer
    - Model different factors separately, as far as possible
    - Combine deterministic and probabilistic performance models
  - Gaining insight from measurements
    - Use performance monitoring agents
    - Define services for obtaining performance information (e.g. application performance profiles)

Basic performance model

A simple model for application execution time:

$$Ttime(app,plat,t) \approx Otime(app,plat,t) \text{ plus } Mtime(app,plat,t)$$

Where:

- *Ttime* means "Total execution time of application"
- *app* means "parameters that characterize the application"
- *plat* means "parameters that characterize the platform"
- *t* means "time" as in "time of day"
- *Otime* means "execution time of application with Overhead produced by external load at *t*"
- *Mtime* means "*pure*" Metasystem overhead time

app

- At least, *app* should include factors such as
  - Number of processes
  - Memory usage characterization parameters
  - Computation characterization parameters, e.g. number of floating point operations
  - Communication characterization parameters, e.g. number of messages and bytes transmitted
  - I/O characterization parameters, e.g., number of I/O operations and bytes transferred

plat

- At least, *plat* should include factors such as
  - Number of nodes
  - Memory characterization parameters (size)
  - Computation characterization parameters, e.g. in terms of asymptotic performance parameters $(r_\square, n_{1/2})$
  - Communication characterization parameters, e.g. in terms of asymptotic performance parameters or *LogP* parameters
  - I/O characterization parameters

Otime(app,plat,tod)

We can model *Otime* using a probabilistic approach

$$Otime(app,plat,t) = f(Eload(t), plat, app)$$

or we can combine probabilistic and deterministic models

$$Otime(app,plat,t) = f(Eload(t), Itime(app,plat), app)$$

Where:

- *Eload* means "external load"
- *Itime* means "execution time if platform were Isolated"
- *f* can be estimated by solving, for example, a queuing model

Otime(app,plat,tod)

- If the platform is isolated (I1), then

  Otime(app,plat,t) = Itime(app,plat)

- Models to estimate *Otime* can be probabilistic in I2 platforms
- The metasystem should provide services for helping estimate *Otime*

Eload(t)

- Returns a list of measured factors representing resource usage
- Needs a set of performance monitoring agents to make the measurements
- The metasystem should provide services for obtaining *Eload*

Eload(t)

- The external workload can be characterized by using
  - Averaging
  - Specifying dispersion
  - Single-parameter histograms
  - Multiparameter histograms
  - Principal-component analysis
  - Markov models
  - Clustering

Mtime(app,plat,t)

- *Mtime* gives execution time of the metasystem
- Models to estimate this factor can be probabilistic
- Effect on the design of the metasystem:
  - services to estimate *Mtime* should be provided

Itime(app,plat)

- Models to estimate this factor may be deterministic (e.g. *LogP*, $(r_\infty, n_{1/2})$ etc.)
- For more accurate estimations, the user should be able to obtain performance profiles of previous executions
- Effect on the design of the metasystem:
  - services get *plat* from the metasystem
  - services to specify *plat* as a minimum requirement
  - services to obtain application performance profiles

Metasystem Performance

Metasystem Performance

- Back to basics:
  - What is the performance information we should provide in commands like "top"?
  - Does it make sense to characterize/model the performance of a whole metasystem?
  - Which performance metrics are appropriate to describe a metasystem?
  - Benchmarks: Can we develop benchmarks for metasystem comparison?

Platform performance

- A platform is a subset of machines that belong to the metasystem, on which we can run parallel applications
- A metasystem may be composed of a set of execution platforms, not necessarily a partition
- Execution platforms may be modeled by benchmarks that estimate *plat* parameters
- Metasystem performance may be characterized in terms of performance measurements obtained from the platforms (best, average, etc.)

Impact on existing networks

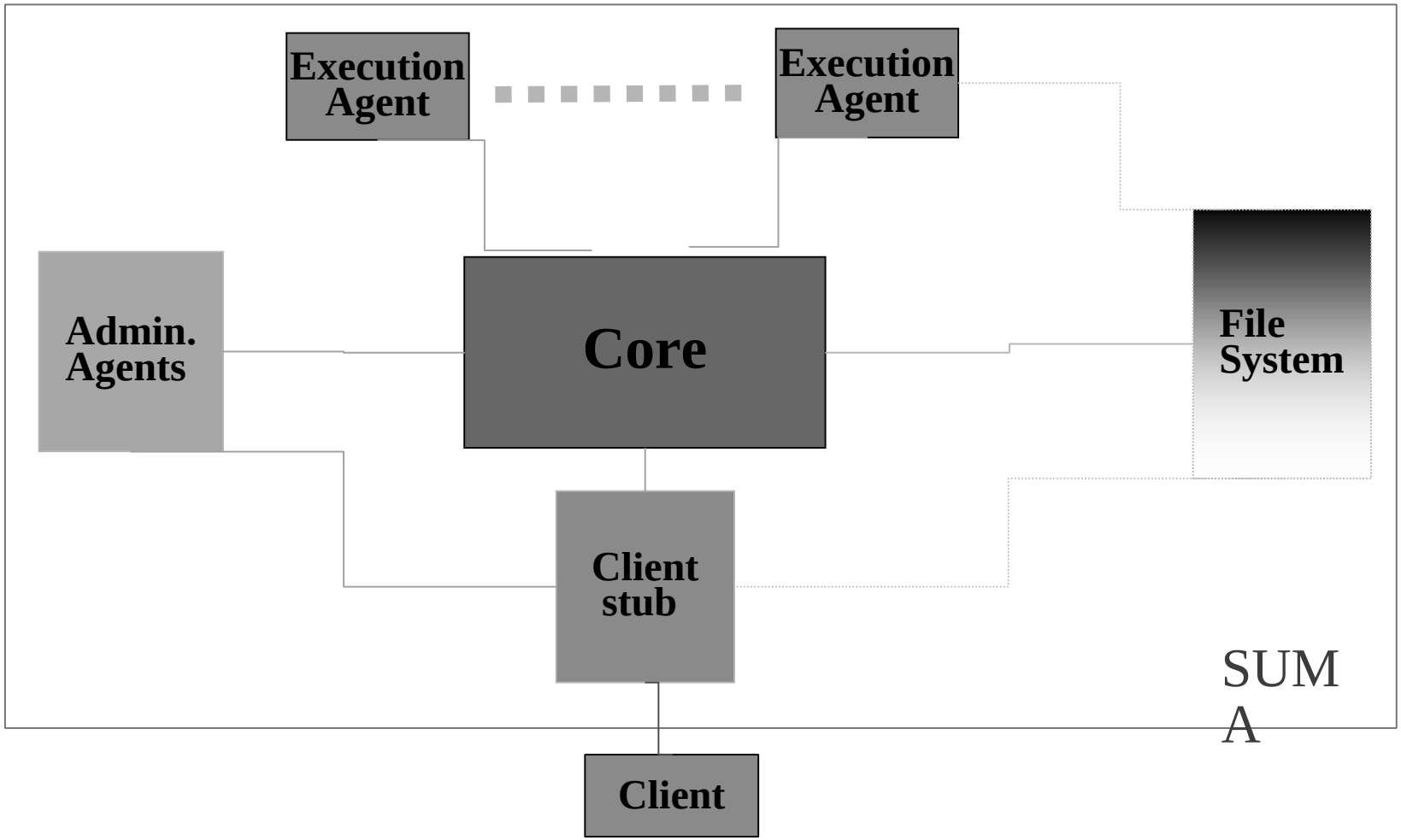
- Distributed execution model on networks designed for a client-server execution model
- Communication between processes as well as program and file load will affect normal function of networks
- Reducing the impact
  - Execution of parallel programs on isolated platforms
  - Metasystem communication channels different from best-effort channel (still, metasystem applications may interfere with each other)

SUMA

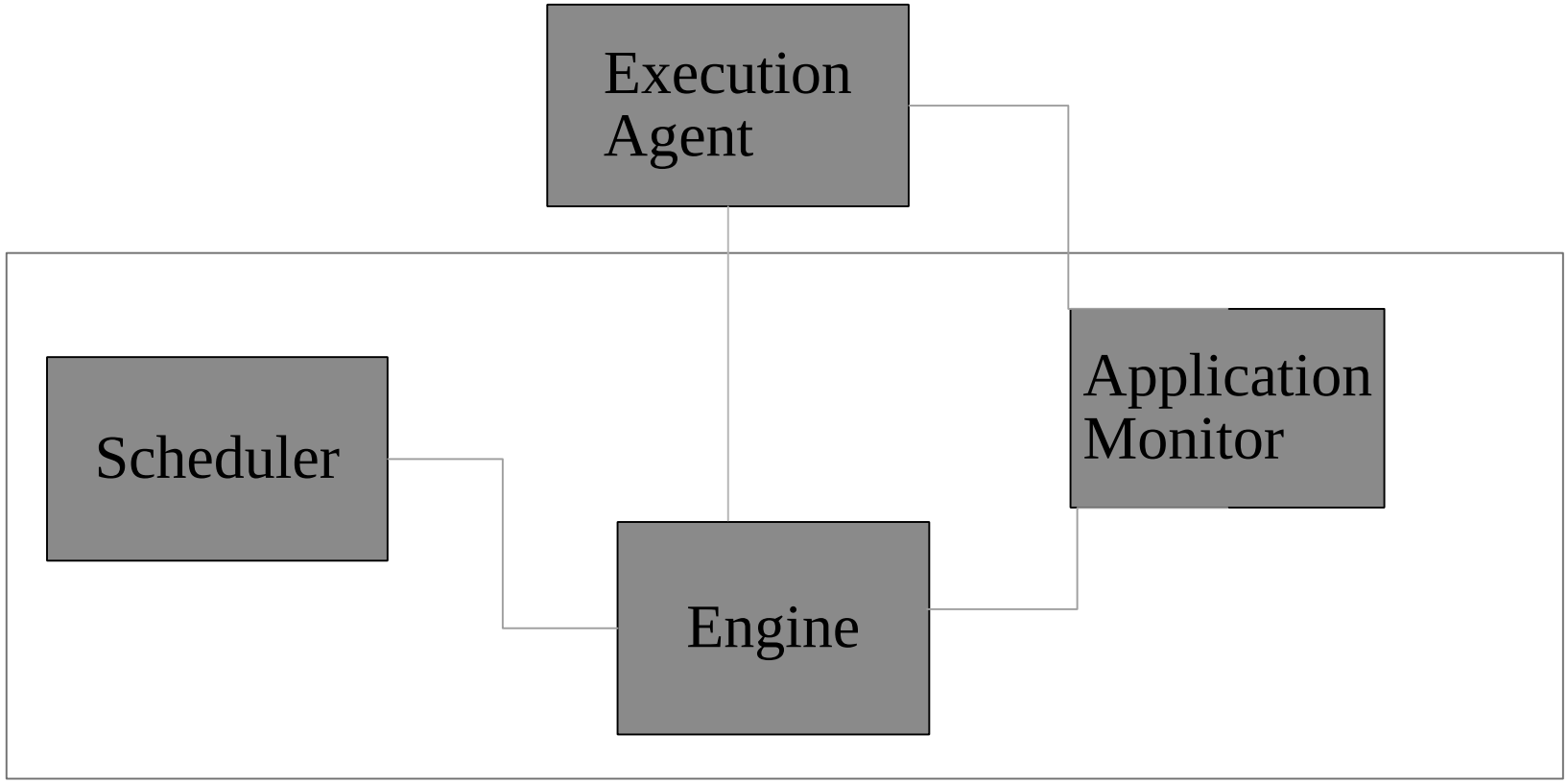
Scientific Ubiquitous Metacomputing Architecture

- Executes Java bytecode with MPI calls
- Built on top of JVM and Corba
- Executes parallel programs in predefined platforms
- Isolation levels are defined for platforms

SUMA design


Execution Agent
Execution Agent
Admin. Agents
Core
File System
Client stub
SUMA
Client

SUMA core



Application Performance Analysis

- Application performance profiles
  - Services to obtain performance profiles
  - SUMA metrics and formats for post-mortem performance profiles
  - More ambitious:
    - post-mortem performance profiles of parallel applications, for space-time diagrams like in ***upshot***
    - dynamic performance profiles like in ***paradyn***

Mtime(app,plat) in SUMA

- *Mtime(app,plat)* is SUMA Core performance
- Particular features
  - SUMA Core is (going to be) a distributed application
  - its performance highly depends on Corba performance
  - a combination of performance monitoring and a queuing model can be used to model *Mtime*

Itime(app,plat) in SUMA

- *Itime(app,plat)* is the application execution time on a isolated SUMA execution platform
  - Java Virtual Machine performance
  - Computation performance
  - Communication performance
  - I/O performance
    - if remote I/O is used the execution platform is not I1

Otime(app,plat,t) in SUMA

- Estimated by specialized SUMA clients
- Performance monitoring agents running on the Execution Servers and SUMA core provide the *Eload* parameters
- *plat* is provided by SUMA services
- *app* is provided by the user (probably with the help of automatic tools)

Conclusions

- SUMA: dual challenge
  - model what we design
  - design what we can model
- Main effects of this approach on SUMA design
  - Predefined parallel platforms (computer clusters within SUMA)
  - Isolation levels
  - SUMA clients designed to help estimate performance