SUMA

Scientific Ubiquitous Metacomputing Architecture

SUMA: A Scientific Metacomputer

Cardinale, Yudith
Figueira, Carlos
Hernández, Emilio
Baquero, Eduardo
Berbín, Luis
Bouza, Roberto
Gamess, Eric
García, Pedro

Universidad Simón Bolívar

MOTIVATION

- In 1999, a couple of projects from USB received funding from a strategic alliance between the government and the Oil industry (Agenda Petróleo): one from Chemistry and another from Geophysics.
- We wanted to build a system that
  - Provides uniform access, from researchers' desktop computers, to campus distributed heterogeneous resources
  - Efficiently supports *high level* scientific programming
  - Offers evolved services (performance, fault tolerance, specialized clients)

BASIC FEATURES

- Execution architecture composed by (heterogeneous) clusters, workstations, specialized hardware, loosely interconnected
- Executes Java byte code, both sequential and parallel
- Support for fault tolerance and recovery
- Provides for efficient execution and performance modeling
- Built on standard, flexible, and portable platforms: Java, CORBA, OO approach

CONTENTS

- Execution basics
- System architecture
- SUMA Components
- Performance
- Fault Tolerance
- Parallel execution
- Current prototype
- Conclusions and future work

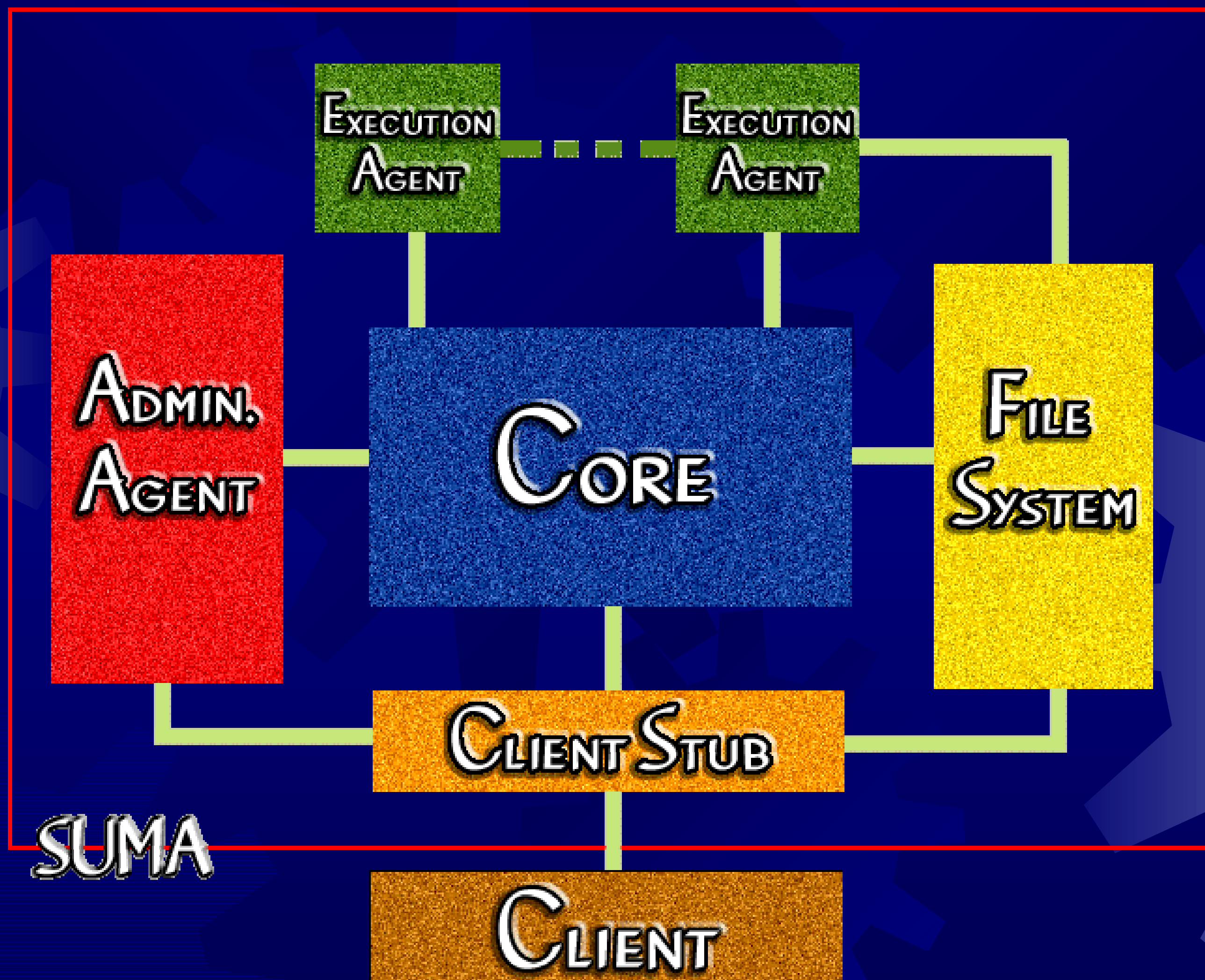
Execution basics

- Two execution modes:
  - *On-line*: a program is supplied to SUMA (e.g. *SUMAjava* main.class). Input and output are redirected to the client machine from the remote node
  - *Off-line*: all classes and input files needed by the application are packed and delivered for execution. Results can be obtained later
- A number of execution attributes can be provided along with the program. For instance, scheduling constraints, classes and data files to be preloaded, etc.

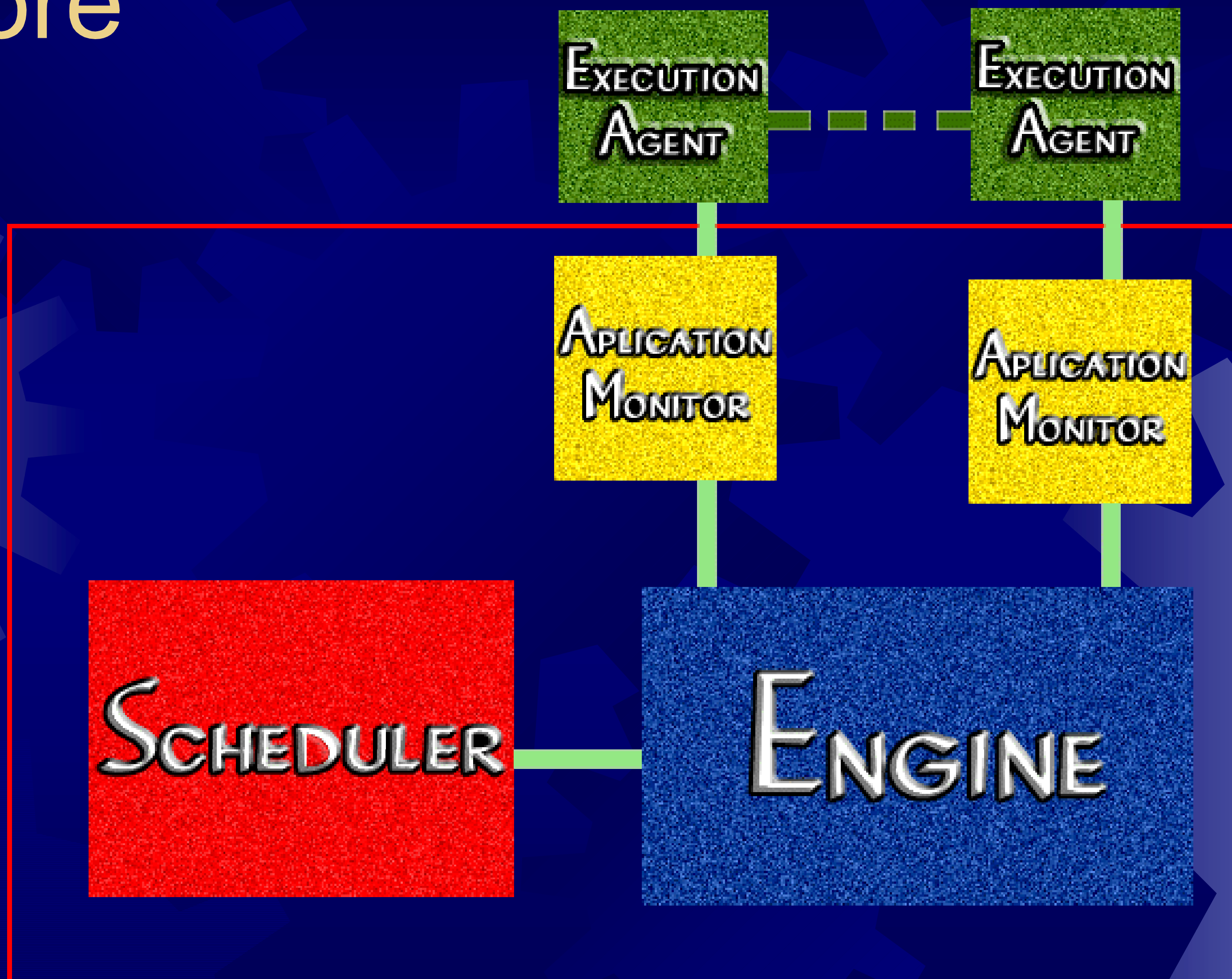
Execution basics (inside SUMA)

- Once the *main class* of the program for execution is given to SUMA from a client machine
  - transparently, SUMA finds a *server* (i.e., a cluster or machine) for execution and sends a request message to that *server*.
  - an *execution agent* at the designated *server* starts execution of the program, dynamically loading the required classes and input data from the client, as well as sending back the output.
  - in case of off-line jobs, output is kept in SUMA until requested by the user.

System architecture



Core


Execution Agent
Execution Agent
Aplication Monitor
Aplication Monitor
Scheduler
Engine

SUMA Components: Core

Engine:

- Coordinates execution
- Receives *Execution Unit* object from client stub
- Checks for permission
- Asks scheduler for suitable *server*
- Delivers *Execution Unit* to designated *server*
- Interacts with Application Monitor
- Handles results, in case of off-line jobs

SUMA Components: Core

Scheduler:

- Obtains status and load information from the *servers*.
- Responds to the Engine requests, based on the applications' requirements.
- Maintains load balance between the *servers*.

SUMA Components: Core

Application Monitor:

- Consists of a *Coordinator* and several *Application Monitor Slaves*
- Receives status information from *Execution Agents* (crash, exit, ...)
- Provides information for implementing
  - Fault Tolerance (based on checkpointing and recovery)
  - Performance modeling and profiling

SUMA Components: Execution Agent

- One per *server*, concurrent.
- Registers itself in the *Resource Control*
- Executes programs
  - Receives *Execution Unit* from the *Engine*.
  - Starts execution, possibly loading classes and files dynamically from the client.
  - Sends result to the client.
- For a parallel platform, the *Execution Agent* plays the role of the *front end*.

SUMA Components: Administration

- Resource control:
  - Used for registration of SUMA resources, i.e., *servers*.
  - Keeps static and dynamic information about the *servers*, such as memory size, available libraries, load, etc.
- User control:
  - Used for user registration.
  - Allows user authentication.

SUMA Components: Client stub

- The client stub is a library for SUMA clients implementation.
- Provides services for *on-line* and *off-line* execution, retrieving results and performance profiles.
- Creates and delivers the *Execution Unit* and *Information Unit*.
- Serves callbacks from *Execution Agents*.
- Two types of clients: User and Administrator.

Performance

SUMA optimizations

- Keep pool of processes at *servers*, with pre-loaded virtual machines
- Remote class loading and *pre-loading*
- Compiling to native code at *servers*
- Others (see *Parallel execution*)

Performance

Application performance feedback

- Provides the user with relevant information concerning performance of application execution (e.g., architecture, etc.)
- Allows for performance tuning, architecture selection, etc.

Fault Tolerance

- At two levels
  - SUMA level, by replicating SUMA components.
  - Execution *server* level, by providing checkpointing and recovery, both sequential and parallel.

Parallel execution

- Parallel platforms in SUMA are *predefined* clusters.
- A parallel platform must provide:
  - MPI
  - Numerical libraries
- Support for executing parallel Java applications with calls to mpiJava.

Parallel execution: services

- ***mpiJava*** is a group of Java classes that allow us to call a native implementation of MPI (1.1) from Java.
- ***plapackJava*** is a set of Java classes that allows users to call the functions of PLAPACK from Java
- ***plapackSUMA*** and ***mpiSUMA*** are implementations of the libraries above using Cygnus Java compiler

Parallel execution: experiment

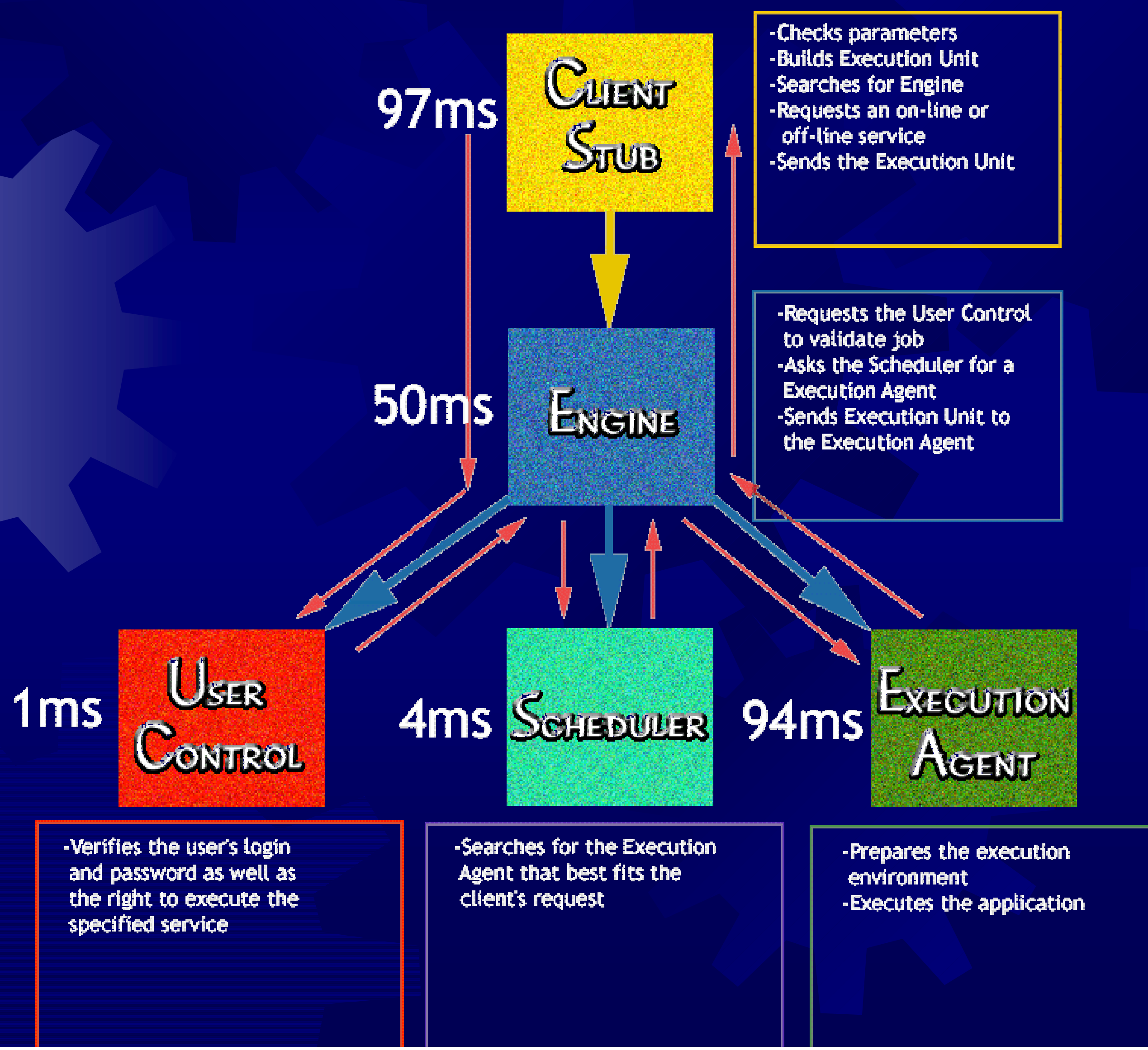
- Results of comparing execution of PLAPACK interfaces (Java and C implementations)
- The experiment consists of solving a linear algebra problem (LU factorization) on a cluster of 8 Pentium II (400 MHz) with 512 Mbytes of RAM, connected with 100 Mbps Ethernet

Size (bytes)	*50*	*100*	*500*	*1000*	*4000*	*6000*
PLAPACK+gcc (s)	0.49	0.87	5.94	13.63	242.72	674.03
plapackJava (s)	0.59	1.01	6.18	13.75	243.45	674.78
plapackSuma (s)	0.52	0.89	6.01	13.65	242.87	674.15

Current prototype

- Centralized Core, public domain CORBA implementation (JacORB 1.14), JDK 1.2, Cygnus compiler.
- Implementations of mpiJava on LAM, for Linux.
- Straightforward scheduling and fault tolerance.
- Runs on Solaris and Linux

Experiments and results



Conclusions and future work

- Basic, expandable, flexible platform for executing Java bytecode, with support for efficient parallel execution
- *Long list of future developments.* We will focus on fault tolerance, and performance tuning and modeling