Προσομοιώσεις Εξέλιξης Φωτιάς σε Κατανεμημένα Συστήματα με Χρήση Πρακτόρων

Thomas Diamantis

Grid Computing for Fire Evolution Simulation

University of Thessaly
Volos

Thomas Diamantis
Department of Computer &
Communication Engineering
University of Thessaly

Thesis supervisors:
Vis. Assist. Prof. Panagiota Tsompanopoulou
Prof. Elias Houstis

Volos, 15 July 2005

Typesetted by the author in LaTeX2$_\varepsilon$.

Contents

List of Figures

List of Tables

Abstract

This work deals with Fire Evolution Simulator (FDS), a software package developed by NIST, USA, and especially with the efforts to grid-enable the original code using agents.

Chapter 1 contains a brief overview of the term “Grid Computing”. In this chapter, we try to give various definitions of the above term, specify the architectures used and outline current as well as with future trends of the research conducted on the area. We finally present as an example the DEISA project, a european effort on grid computing.

Chapter 2 is devoted to middleware. In this chapter we try to define two entities similar to each other, agents and actors and give some examples of platforms used to develop each one. Finally, there is a description of gLite, a grid middleware alse developed in Europe.

Having finished with the theoretical backround, we examine FDS itself in Chapter 3. Namely, we present its structure (input - output files) and the technologies used for its various versions (original code in Fortran with MPI extensions, wrapped with C and Java code for the agentised case).

Finally, in Chapter 4 we examine the results gathered from the experiments performed in our cluster and our local “grid”. These experiments intend to measure the performance of FDS and proove the validity of the agentised code.

CHAPTER

1

Grid Overview

Computational grids as an idea is not something novel. It began as "networked operating systems", in the 70s, returned as "distributed operating systems" in the late 80s and early 90s, then became "heterogeneous computing", "parallel distributed computing", "metacomputing" and finally "computing on the Grid" [38].

All these terms are not necessarily antonyms, but still there exists several differences between today's understanding of grids and older distributed networking: grids focus on site autonomy, involve heterogeneity and are a more generalized concept than just computers and networks. Ian Foster proposed [25] a three-checkpoint list to identify whether a system should be called a grid or not:

1. a grid should coordinate resources that are not subject to centralized control.
2. such a coordination should be done using standard, open, general-purpose protocols and interfaces.
3. the purpose of the above should be to deliver nontrivial qualities of service.

1.1 Definitions

Various definitions have been proposed to describe the term "The Grid". During the mid 90s, "the grid" was used to denote

- *a distributed computing infrastructure for advanced science and engineering* [26].

In 1998, Carl Kesselman and Ian Foster attempted another definition in the book "The Grid: Blueprint for a New Computing Infrastructure":

- *A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities* [25].

Others [17] claim - in a more simple way - that grids are

- *collections of computational and data storage resources linked by communication channels for shared use.*

Yet another definition [29] is that a grid is

- *a set of tools and technologies that allow users "easy" access to resources and applications*

Of course, all the definitions mentioned above are rather complementary than competitive.

If we now try to examine grids from an end-user perspective, they are used to provide various types of services such as *computational services* (execution of applications on distributed computational resources), *data services* (scalable storage and access to data sets stored in different locations accumulating in petabytes of mass storage), *application services* (application management and transparent access to remote software and libraries) and *information services* (extraction and human readable representation of data using the services above) [18].

Generally speaking, computational grids are a lot analogous to electrical power grids [28]. The latter provide lowcost and reliable access to a standardized service (electricity), allowing both individuals and industries to take for granted the availability of cheap and reliable power. Similarly, computational grids do almost the same for computational power.

1.2 Why use computational grids?

Considering the continuous - and sometimes astonishing - progress that has been made on computer performance during the last decades, one could raise the question: *"What do we need computational grids for? Why should we connect together machines which are powerful enough on their own?"*.

At first glance, such a concern would look rational. Today, the PC owned by a 10-year-old child is stronger and faster than a 10-year-old supercomputer. Nevertheless, such a PC is still far from adequate for computing the results of complex models in real time. A well-known example of such kind is weather forecasting. Given the intricacy of the predicting model, one would aquire tomorrow the results for yesterday's conditions.

Computational power is believed to increase even more through revolutionary breakthroughs in a wide range of areas [28]:

Technology innovations in terms of VLSI technology and microprocessor architecture. Moore's law is still holding, remember? [12]

Increase in demand-driven access to computational power Given the existence of high-end systems in coordinance with - hypothetical - mechanisms allowing reliable and transparent access to them, an increase in computational power is a realistic prediction.

Increased utilization of idle capacity A lot of efforts have been made these days towards full utilization of PCs and workstations resulting to an increase in peak computational capacity. See SETIHome [14] for instance, which is a scientific experiment that uses Internet-connected computers in the Search for Extraterrestrial Intelligence.

Greater sharing of computational results Effective sharing is limited in our days but in the future it may be not, as more and more scientists are trying to adopt a more holistic approach to computation.

New problem-solving techniques and tools New approaches and techniques allow for a more efficient - in terms of computation - problem solving.

This is where grids join the game. They will provide the infrastructure to make the innovations mentioned above feasible, at first, and then available to more and more people around the globe.

1.3 Architecture

Grid computing assumes that any potential participants should be able to establish communication and sharing relationships. The essential condition for this is interoperability, which, in a networked environment such as grids, is further translated to common protocols. Thus, grid architecture is mainly a protocol architecture, with protocols defining the basic mechanisms by which users and resources negotiate, establish, manage, and exploit sharing relationships [26].

1.3.1 Components

Foster and others suggest [26] a five-layered protocol architecture for grids (see Figure 1.1):

Fabric this layer provides the resources to be shared. These resources can vary from a logical entity, such as a distributed file system, a single PC or a computer cluster, to storage devices and database or even special scientific instruments such as a radio telescope or heat sensors [26, 18].

Connectivity this layer offers core communication and authentication services. Communication involves transport, routing and naming. It is assumed that grids use the existent protocols (i.e., TCP for transport, DNS for naming), but this does not mean that in the future the need for new protocols will not emerge.

Resource based on the protocols defined by the previous layer, Resource layer offers services such as information registration and discovery (negotiation, initiation), remote process management (monitoring, control) and aspects of Quality of Service (QoS) like accounting and payment of sharing operations.

Collective this is the user-level layer which includes directory services that allow participants to discover the existence and/or properties of resources, co-allocation, scheduling, and brokering services that allow participants to request the allocation of one or more resources and perform the scheduling of tasks, monitoring and diagnostics services, data replication services whose purpose is to support the management of storage resources to maximize data access performance, grid-enabled programming systems, workload management systems and collaboration frameworks, software discovery services and others.

Application the final layer consists of the user applications that operate on grids. These applications are often developed using grid-enabled languages and utilities such as HPC++ or MPI.

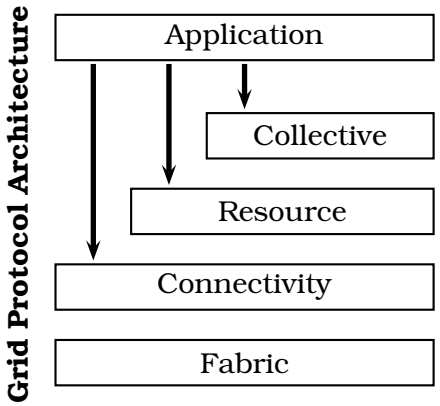


Figure 1.1: Five-layered grid architecture.

One can further tie the Connectivity and Resource layers in one and construct a shorter list of essential grid components (see Figure 1.2): [18]

- Grid fabric
- Core grid middleware
- User-level grid middleware
- Grid applications

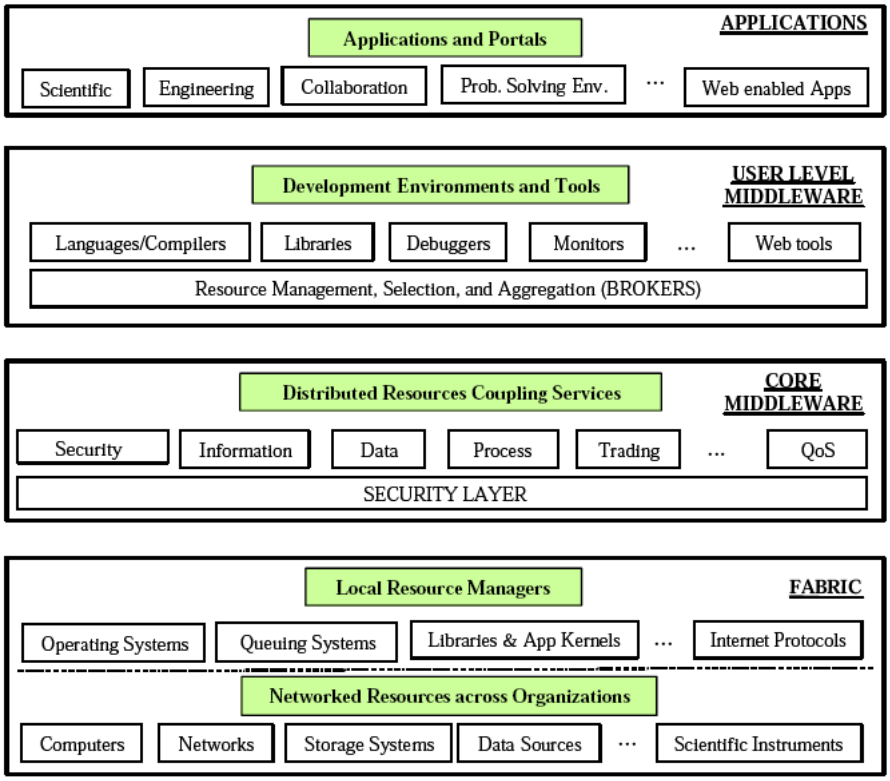


Figure 1.2: Grid components.

1.4 Future Work and Research for the Grid

It is obvious that in a field of rapid progress such as informatics it is difficult to "predict the future", thus foreseeing what will become the ruling Grid approach in a few years is not an easy task. However, one can distinguish some tendencies, one of which is the growing interest in the use of Java and Web services [18]. This happens because the language itself incorporates some of the most important demands of grids such as heterogeneity and security, not to mention that it was designed from the beginning as "a language for the internet".

Of course there are still issues to be addressed, problems to be solved before grids can evolve to a next level. Some of those challenges include [28]:

The Nature of applications History has taught us that changes in capabilities (i.e., from a workstation to a grid) may lead to new ways of using computers. Hence, research is required to explore what is possible to be done, not only in the "traditional" scientific and engineering domains but in other areas as well, such as business art and entertainment.

Programming models and tools Grids introduce new models and architectures. As a consequence, new techniques are required for mapping existing or new algorithms on grid environments.

System architecture Systems supporting grid environments have to be scalable

and often have to comply with contradicting requirements such as simplicity (in order to achieve broad deployment) as well as with complexity (so that a wide variety of complicated and performance-sensitive applications can be supported). All the above are not yet satisfied by existing technologies.

Algorithms and problem solving methods New algorithms are needed as grids differ considerably from single and even multiprocessor and parallel computing systems.

Resource management One of the key issues of grid systems is sharing of common resources. Therefore, the problem of managing these resources effectively, arises.

Security Current network security usually gives more emphasis on traditional client-server interactions. Grids, however, involve more entities and more complex activities, introducing new challenging security problems.

Instrumentation and performance analysis Performance is many times critical to grid applications and this makes technologies for collecting, analyzing and explaining performance data of vital importance.

End systems Today's end systems are designed to operate with interfaces and operating systems developed to read and write on slow disks. Grids call for next-generation high-performance networking, emerging to new approaches to both low level (operating system, network interface) and higher level areas (new applications for networked computers).

Network protocols and infrastructure Last but not least, high bandwidths and performance assurances that grid applications require are likely to render current network protocols and infrastructure insufficient, thus making the proposal of new technologies for transport, switch, route, and manage network flows essential.

1.5 DEISA - A grid example

DEISA - Distributed European Infrastructure for Supercomputing Applications [3] is a european effort (still under development) to connect several supercomputers operating in Europe to an intercontinental grid. Leading national supercomputing centers have formed this consortium that currently deploys and operates a persistent, production quality, distributed supercomputing environment with continental scope. The purpose of this research infrastructure is to enable scientific discovery across a broad spectrum of science and technology, by enhancing and reinforcing European capabilities in the area of high performance computing. This becomes possible through a deep integration of existing national high-end platforms, tightly coupled by a dedicated network and supported by innovative system and grid software.

Indeed, DEISA is structured as a layer on top of the national supercomputing services, and coexists with them. This infrastructure addresses the computational challenges that require the coordinated action of the different national supercomputing environments and services for both efficiency and performance.

DEISA provides leading scientific users with transparent access to a European pool of computing resources. The coordinated operation of this environment is tailored to enable new, ground breaking applications in computational sciences.

The DEISA Consortium follows a top-bottom strategic approach for the deployment and the evolution of the infrastructure. The Consortium does not commit "a priori" to any specific technology. Its technology choices are fully open, and they follow from the strategic requirements and the operational model of the DEISA virtual organization. A few very basic strategic requirements have determined the major initial choices made for the deployment of the infrastructure:

- The necessity of fast deployment of a persistent, production quality supercomputing infrastructure with continental scope.

- The coexistence of the European infrastructure with the national services, which requires reliability and non-disruptive behavior.

- User transparency (users should not be aware of complex grid technologies) and applications transparency (minimal intrusion on applications, which, being part of the corporate wealth of research organizations, should not be strongly tied to an IT infrastructure).

1.5.1 DEISA Architecture

The integration of national resources inside the DEISA supercomputing grid operates at two levels (see Figure 1.3):

- An inner level, dealing with the deep integration and strongly coupled operation of similar, homogeneous platforms. Here, national IBM AIX clusters are glued together to constitute a distributed European supercomputer, called "the AIX super-cluster".

- An outer level, dealing with a looser federation of heterogeneous supercomputing resources. This constitutes a heterogeneous grid of supercomputers and super-clusters. Indeed, this heterogeneous supercomputing grid includes all the leading platforms in Europe exhibiting different technologies from different vendors (IBM, SGI, NEC). In this context, the AIX super-cluster is seen as a single platform.

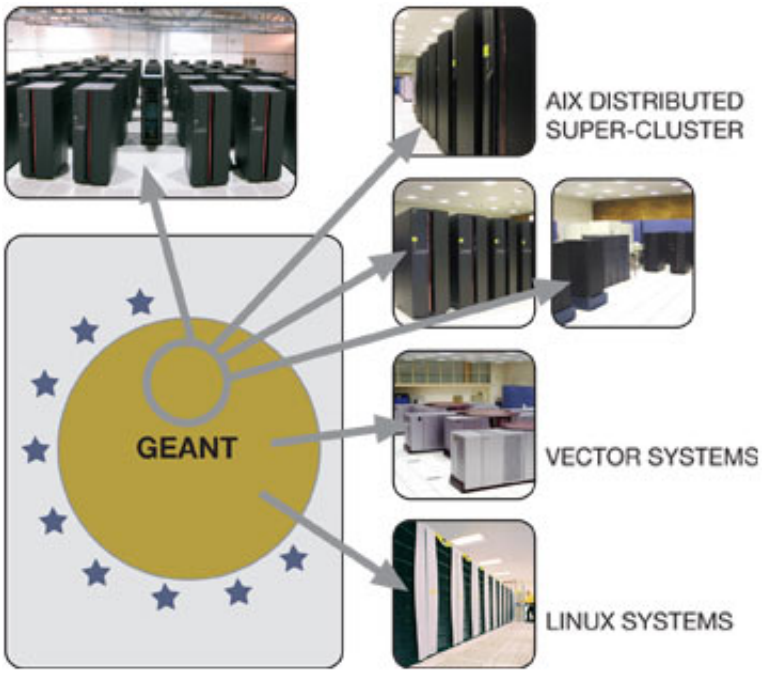


Figure 1.3: DEISA Architecture.

The DEISA network

DEISA uses an internal network provided by GEANT[1] and the National research networks(NRNs) that connects the supercomputers and offers reserved bandwidth, This internal network exists, of course, in addition to the standard Internet connectivity that each national supercomputer centre offers.

The AIX super-cluster

The phase 1 of the AIX super-cluster involves four IBM Power4 platforms:

- **FZJ-Julich (Germany)** P690 (32 processor nodes) architecture, incorporating 1312 processors. Peak performance is 8.9 Teraflops.
- **IDRIS-CNRS (France)** Mixed P60 and P655+ (4 processor nodes) architecture, incorporating 1024 processors. Peak performance is 6.7 Teraflops.
- **RZG{Garching (Germany)** P690 architecture incorporating 896 processors. Peak performance is 4.6 Teraflops.
- **CINECA (Italy)** P690 architecture incorporating 512 processors. Peak performance is 2.6 Teraflops.
- **CSC (Finland)** P P690 architecture incorporating 512 processors. Peak performance is 2.2 Teraflops.

[1] GEANT is an effort to create a multi-gigabit pan-European data communications network, reserved specifically for research and education use [6].

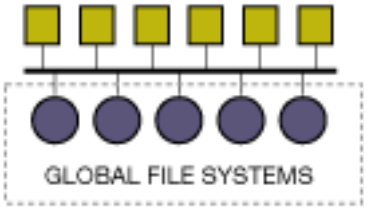


Figure 1.4: Cluster file system.

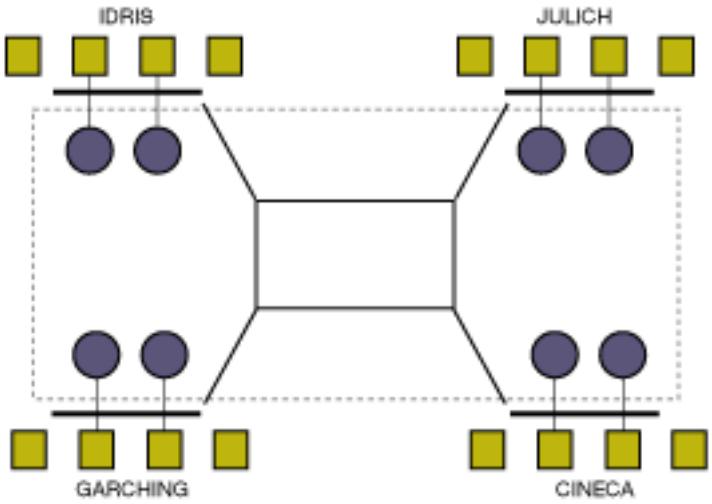


Figure 1.5: Grid filesystem.

The fundamental integration concept in this area is transparent access to remote data files via a global distributed file system. This is a natural wide area network extension of standard cluster architectures, and this is why we speak of a super-cluster. Each one of the national supercomputers listed above is a cluster of several autonomous computing nodes, linked by a high performance network. Data files are not replicated on each computing node, they are unique and shared by them all (see Figure 1.4). A data file in a global file system is "symmetric" with respect to all computing nodes, and can be accessed with equal performance from all of them. Therefore, a user does not need to know (and does not know in practice) on which set of nodes his application is executed.

The IBM AIX systems listed above are running IBM's GPFS (Global Parallel File System) [7] as a cluster file system. Recently, IBM has incorporated wide area network functionality in GPFS, enabling the deployment of distributed global file systems (see Figure 1.5). This is the basic integration technology of the AIX super-cluster.

An application running on one site can access data files previously "exported" from other sites as if they were local files. Therefore, it does not matter in which site the application is executed, and applications can be moved across sites transparently to the user.

It is important to emphasize that the main issue here is high performance remote access needed for high performance computing, and this is what GPFS is supposed

to provide. On the DEISA infrastructure applications are observed to utilize the full 1 Gb/s bandwidth of the underlying network when accessing remote data via GPFS. As another example, on the 30 Gb/s TeraGrid network in the USA, GPFS runs at 27 Gb/s in remote file accesses. This software is therefore capable of taking full advantage of underlying high performance networks. The ultimate performance of the AIX super-cluster is therefore strongly linked to the performance of the next generation network infrastructures in Europe.

The full heterogeneous supercomputing grid

The DEISA Supercomputing Grid involves a number of DEISA partners that contribute a significant amount of the national supercomputing resources (of the order of 10% or more) to a globally managed European resource pool. The leading supercomputing platforms in Europe participating to the DEISA resource pool are:

- The AIX super-cluster (FZJ, RZG, IDRIS, CINECA, CSC) discussed above, with an aggregated computing power close to 25 Teraflops.
- BSC (Barcelona, Spain): 4564 Power PC processor IBM Linux system, 9 TB memory space, 233 TB disk space. Peak performance is 40 Teraflops.
- HLRS (Germany): NEC SX8 vector supercomputer, 576 processors, 9.2 TB memory space, 180 TB disk space. Peak performance is12.67 Teraflops.
- LRZ (Germany): Linux cluster with more than 500 CPUs (above 2.7 Teraflops peak performance) (see below for the planned evolution of this platform)
- SARA (The Netherlands): SGI Altix 3700 Linux system, 416 Itanium-2 processors , 832 GB main memory, peak performance 2.2 teraflops.
- ECMWF (International organization): Two 690+ clusters with 68 32-way nodes each, with an aggregated performance of 33 Teraflops peak.

At this level, the objective is to federate supercomputing environments in a supercomputing grid that will include, in addition to the supercomputing platforms mentioned above, a number of data management facilities and auxiliary servers of all kinds. The DEISA Supercomputing Grid can be seen as a global architecture for a virtual European supercomputing centre.

1.5.2 Operation and services

The AIX super-cluster is supposed to run bigger and more demanding applications than the ones that can be run today on each national cluster. A common approach of doing this would be to “grid enable” the application so that it can run on more than one platform. However, this strategy - that requires a modification of the application - does not really work for tightly coupled parallel applications.

DEISA adopts a different strategy, based on load balancing the computational workload across national borders. Huge, demanding applications are run by

reorganizing the global operation in order to allocate substantial resources in one site. They are therefore runs "as such" with no modification. This strategy only relies on network bandwidths, which will keep improving in the years to come.

The other benefit of the AIX super-cluster comes from the possibility of transparently sharing data through GPFS. European data repositories that require frequent updates - like bio-informatics databases, for example - can be established in one site and accessed by all the others.

As a result, the AIX super-cluster provides to end users most of the benefits of a unique, 20 teraflops supercomputer, in a transparent way.

Additionally, the DEISA grid provides services like support for workflow operations (those that need to "visit" successively different computing resources to accomplish a complex simulation) and portal and web interfaces (in order to enhance the user adoption of sophisticated supercomputing infrastructures).

Finally, as an exception to the "do-not-grid-enable-the-applications" strategy mentioned before, there is an implementation of a co-scheduling service on the supercomputing grid for those sophisticated multi-scale, multi-physics applications composed of loosely coupled independent software modules that need to exchange information in real time with limited communications overhead. Classical examples are ocean-atmosphere or fluid-structure coupled codes.

Altogether, the DEISA grid has deployed activities on various modern and challenging areas such as material sciences, cosmology, plasma physics, life sciences, and industry.

CHAPTER

2

Middleware

As mentioned in Section 1.3, in order to realize a grid it is necessary, among others to deploy both low- and user-level middleware. The first is needed to provide a secure and transparent access to resources while the latter is used – in coordinance with other tools – to provide the essential infrastructure for application development and the aggregation of distributed resources [18].

2.1 Agents

Similar to grids, there is no universally accepted definition of the term agent. There is a general opinion that autonomy is essential, but beyond this, there is little agreement. Part of the difficulty in defining an agent is that many of its attributes are of different significance for different applications. For instance, some times the ability of an agent to learn by experience is crucial and some times not only is it trivial, but it may be unwanted as well; nobody would like an agent-based air-traffic control system whose agents would modify their behavior at run time.

A definition adopted by Wooldridge and Jennings [39] states that

- *an agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objectives*

Yet another definition focusing on agent attributes describes agents as [19]

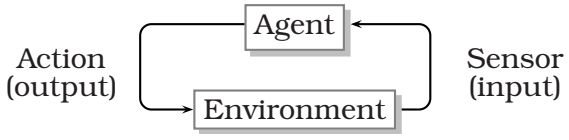


Figure 2.1: Agent schematic representation.

- *a software component that is autonomous (has a degree of control over its own actions), proactive (does not only react in response to external events but also exhibits a goal-directed behavior and, where appropriate, is able to take initiative) and social (it is able to, and need to, interact with other agents in order to accomplish its task)*

An agent will typically sense its environment (by physical sensors in the case of agents situated in part of the real world or by software sensors in case of software agents) and decide what to do from a specific repertoire of actions available to it. Of course the agent has not overall control of the environment. Given the environment's non-deterministic nature, the action performed by the agent will probably modify it but it will not necessarily have the desired effect.

Recognizable entities that can be viewed as agents are control systems (like a thermostat controlling the temperature of a room) and daemons (i.e., background Unix processes) which monitor a software system and perform actions accordingly.

A special category of agents are mobile agents. These are agents which can move around an electronic network [34] just like a robot would move around in real world. For example, programs roaming a network to collect business-related data in order to help users to buy goods, or implement platform-independent code-on-demand can be viewed as mobile agents [33].

2.1.1 Agents and grids

Traditionally, grid development focuses on interoperable infrastructure and tools for secure and reliable resource sharing within dynamic and geographically distributed virtual organizations [27]. On the contrary, agent communities have focused more on "brains", trying to develop autonomous problem solvers that can act flexibly in a dynamic environment [27]. Nevertheless, as the scale and ambition of both communities rises, their interests are starting to overlap; grids are trying to become more flexible and agents are trying to be based on a more robust infrastructure which will provide more scalability and security.

Examples (random and non-representative) of using agent-oriented techniques in grid environments are [23], [22] and [30]. The first describes a methodology of resource management for grid computing consisting of agents who are capable of cooperating with other agents to provide service advertisement and discovery to schedule applications that need to utilize grid resources. The second paper introduces a mechanism which provides agent-based self-organization in order to perform complementary load balancing for batch jobs with no explicit execution

deadlines. Finally, the third paper proposes a way of constructing a grid from scratch based on mobile agents.

2.1.2 Agent platforms

Numerous agent-oriented platforms have been developed. Five of them are mentioned below. Additionally, these platforms are rated in terms of security issues, development and standards compliance[1]. The overall rating is shown in Table 2.2.

	PRODUCT	TYPE
1	Bee-gent	Language or environment for agent development
2	JADE	Distributed Agent platform
3	Kaariboga	Language or environment for agent development
4	Voyager	Support software
5	Pro-active	GRID platform

Table 2.1: Some agent platforms.

Bee-gent

Bee-gent (Bonding and Encapsulation Enhancement Agent), is a communication framework based on the multi-agent model [34]. It is a new type of development framework in that it is a 100% pure agent system. As opposed to other systems which make only some use of agents, Bee-gent provides applications with autonomous network behavior by "agentifying" them (i.e., providing an agent interface). It then supports agent-based inter-application communication, facilitating co-operation and problem-solving. Bee-gent achieves this in a flexible and open structured manner, making it well suited to providing for co-operative processing in the advanced network society.

JADE

JADE is an enabling technology, a middleware for the development and run-time execution of peer-to-peer applications which are based on the agents paradigm and which can seamless work and interoperate both in wired and wireless environment.

Fully developed in the Java language, JADE platform tries to comply with the following principles

- **Interoperability** By implementing the standard[2], JADE agents are capable of

[1] Project ratings is a work conducted in 2003 by John Michopoulos, Naval Research Laboratory, USA, (personal communication)

[2] Specifications proposed bu The Foundation for Intelligent Physical Agents (FIPA) which are intended to promote the interoperation of heterogeneous agents and the services that they can represent [5]

interoperating with other agents provided that the latter are also compliant with the same specifications.

- **Uniformity and portability** JADE's APIs are independent of the underlying network and Java version (J2EE, J2SE, J2ME) allowing - in theory - application developers to decide the Java run-time environment at deploy time.

- **Easy to use** The complexity of the middleware is hidden behind a simple and intuitive set of APIs.

- **Pay-as-you-go philosophy** Programmers do not need to use all the features provided by the middleware as those that are not used do not require programmers to know anything about them, neither do they add any computational overhead.

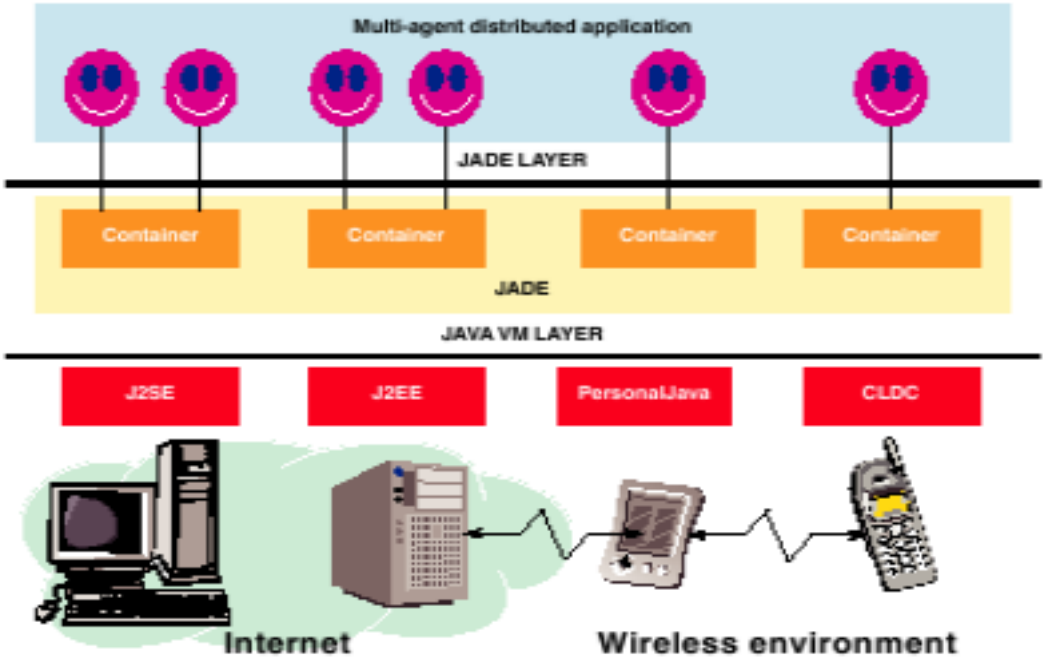


Figure 2.2: JADE Architecture.

Kaariboga

Kaariboga Mobile Agents, Kaariboga, is a free – JAVA – implementation of a framework for mobile agents. Although project development seems to be stalled, Kaariboga intends to provide a platform that can be freely used for experiments and research on mobile agents[10].

	General			Security				Development				Standards		
	OS Independent	Documentation	Mobile agents	Authentication	Data encryption	Authorization	Access restriction	Monitoring	Debugging	RAD	Architecture	FIPA	GLOBUS	Grid services
Bee-gent	Y	Y	Y	1	2	4	4	4	4	3	4	1	4	4
JADE	Y	Y	Y	1	4	2	4	2	2	4	3	1	4	2
Kaariboga	Y	Y	Y	4	4	4	3	4	4	4	4	4	4	4
Proactive	Y	Y	Y	2	2	2	2	2	2	4	2	4	2	1
Voyager	Y	Y	Y	1	1	4	4	4	4	4	4	4	4	4

Table 2.2: Overall ratings of agent platforms. 'Y' (Yes) indicates that the system satisfies the specific criterion. The grades have been assigned from 1 (best,yes) to 4 (worst,no).

Voyager

Voyager is a licensed product from Recursion Software, Inc. It is a standards neutral, 100% pure Java development platform and object request broker (ORB) for distributed computing, designed to speed development and improve the performance and quality of enterprise solutions [34].

It offers (among others) flexible, layered architecture, portability across multiple operating systems, built-in and custom security features, survivability, scalability, reliability, high performance/fast run-time and is compliant with standards [15].

The Voyager Framework includes:

- Voyager Distributed Development Platform (VDDP)
- Plug-In Modules: Security, Transaction, Messaging
- Voyager Management Console

Pro-Active

Proactive is a GRID Java library for parallel, distributed, and concurrent computing, also featuring mobility and security in a uniform framework. With a reduced set of simple primitives, ProActive provides a comprehensive API allowing to simplify the programming of applications that are distributed on Local Area Network (LAN), on cluster of workstations, or on Internet Grids [13].

Proactive is not a pure agent platform. Apart from enabling the development of a mobile and potentially secure agent, it also provides a uniform way to encapsulate remotely accessible objects, threads as asynchronous activities, actors with their own scripts (see Section 2.2) or servers of incoming requests [13].

2.2 Actors

Actors are autonomous reasoning agents [32]. They are small entities, solely defined by their behavior which is further characterized by its response to receiving a message [37]. In analogy to real-life actors, a key attribute of actor communities is concurrency (real humans operate concurrently in everyday life).

Communication between actors is achieved through an - asynchronous - message passing according to a specific scheme. When a message is received by an actor, the actor determines whether it recognizes it as one for which there exists a programmed response. If so, the action associated with the message is performed and a response, if necessary, is relayed back to the sender of the original request.

Actors have a well defined interface, which abstracts its internal state and execution and restricts how it interacts with its environment. Externally, this interface includes ports that represent points of communication for an actor and parameters which are used to configure its the behavior [36]. This abstraction is a central concept in actor-oriented design; internal behavior and state of an actor are hidden behind the actor interface and are not visible externally. This property of strong encapsulation separates the behavior of a component from the interaction of that component with others.

The main difference between actors and agents is that actors mostly react to messages sent by other actors while agents have sensors and decide what to do according to the current environmental status. Actors don't know anything about their surroundings and expect a message from an external source in order to act. Agents do exchange messages with each other but they also keep an internal representation of the environment (at least of what they can "sense") and perform actions as a consequence of the things the "see" or "hear". Finally, agents have a specific target, a goal which drives their reactions while actors do not.

2.2.1 Ptolemy

A project that supports actor-oriented design is Ptolemy. Ptolemy is a framework to model, simulate, rapidly prototype and design heterogeneous and concurrent systems [20, 32]. It would be ideal for applications for which heterogeneity is an important factor such as design of multimedia networks, real-time embedded software, hardware/software codesign, control and call-processing in telecommunication networks, mixed-mode hardware simulation, mapping applications onto heterogeneous multiprocessor systems and mixed signal processing and real-time control [20]. Generally, Ptolemy focuses on systems that are complex in the sense that they mix widely different operations.

There is also focus on embedded systems especially those which blend different technologies together (i.e., analog and digital electronics, hardware and software, electronics and mechanical devices). The term *embedded software* describes software that is meant to be executed on devices that are not computers, such as automobiles, telephones, pagers, consumer electronics, toys, aircraft, trains, security systems, weapons systems, printers, modems, copiers, thermostats, manufacturing systems,

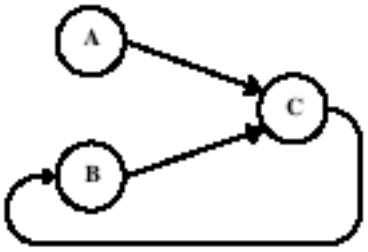


Figure 2.3: General representation of models of computation.

appliances, etc. A main difference between embedded and traditional software is that the first operates on devices that reside in the physical world and hence there are temporal constraints, absent in desktop-oriented systems. As mentioned before actors (i.e., Ptolemy components), act concurrently. Ptolemy implements a variety of models of computation that deal with this concurrency. A - general - graphical representation of models is shown in Figure 2.3, where nodes represent - in most circumstances - actors and arcs represent message exchanges (communication).

Namely, the models implemented in Ptolemy [32] are

- Component Interaction
- Communicating Sequential Processes
- Continuous Time
- Discrete Events
- Distributed Discrete Events
- Discrete Time
- Finite State Machines
- Process Networks
- Synchronous Dataflow
- Giotto
- Synchronous/Reactive
- Timed Multitasking

Describing the above is beyond the scope of this work and the reader can find details in [32].

2.3 gLite

gLite is a European effort to create a lightweight middleware for grid computing which will provide a bleeding-edge, best-of-breed framework for building grid applications tapping into the power of distributed computing and storage resources across the Internet.

2.3.1 Components

gLite consists of various subsystems which provide the essential infrastructure to form virtual organizations - grids.

Computing Element Subsystem

The Computing Element (CE) is the service representing a computing resource. Its main functionality is job management (job submission, job control, etc.). The CE may be used by a generic client: an end-user interacting directly with the Computing Element, or the Workload Manager, which submits a given job to an appropriate CE found by a matchmaking process. For job submission, the CE can work in push model (where the job is pushed to a CE for its execution) or pull model (where the CE is asking the Workload Management Service for jobs). Besides job management capabilities, a CE must also provide information describing itself. In the push model this information is published in the information Service, and it is used by the match making engine which matches available resources to queued jobs. In the pull model the CE information is embedded in a "CE availability" message, which is sent by the CE to a Workload Management Service. The matchmaker then uses this information to find a suitable job for the CE.

Data Management Subsystem

The three main service groups that relate to data and file access are: Storage Element, Catalog Services and Data Scheduling. To the user of the data services the abstraction that is being presented is that of a global file system, with very similar semantics. A client user application may look like a Unix shell which can seamlessly navigate in this virtual file system, listing files, changing directories and accessing the data in the files (access is controlled by Access Control Lists and can be done through the Storage Element).

Accounting Subsystem

The accounting service accumulates information about the usage of Grid resources by the users and by groups of users.This information allows preparation of statistical reports, to track resource usage for individual users, to discover abuses and to help avoid them. Accounting information could be used to charge users for the Grid resources they have utilized. The information available from the accounting service can also be used to implement submission policies based on user quotas or on

resource usage (fair share). In principle it also allows the creation of a real exchange market for the Grid resources and services. The subsequent economic competition should result in market equilibrium, thereby promoting load balancing on the Grid.

Logging and Bookeeping Subsystem

The Logging and Bookkeeping service (LB) tracks jobs in terms of events - important points of job life (i.e., submission, finding a matching CE, starting execution etc.).

The events are passed to a physically close component of the LB infrastructure (local logger) in order to avoid network problems. This component stores them in a local disk file and takes over the responsibility to deliver them further. The destination of an event is one of the *bookkeeping servers* assigned statically to a job upon its submission. The server processes the incoming events to give a higher level view on the job states (e.g., Submitted, Running, Done) which also contain various recorded attributes (e.g., destination CE name, job exit code, etc.). In order to retrieve those details, LB provides a public interface consumer API which is completely passive; it allows querying but no data are pushed beyond the LB server actively.

Information and Monitoring Subsystem

R-GMA (Relational Grid Monitoring Architecture) is based on the GMA from the GGF (Grid Global Forum) [8], which is a simple Consumer-Producer model. The special strength of this implementation comes from the power of the relational model. R-GMA offers a global view of the information as if each Virtual Organization had one large relational database. It is suitable both for information about the grid (primarily to find out what services are available at any one time and then to find details of those services) and for application monitoring. Those with information to share publish it via a “Producer” and those seeking information obtain it via a “Consumer”.

Security Subsystem

The aim of the security system is to be modular (ability to add new modules later), agnostic (modules will evolve), standard and interoperable. A “third-person” view of the security architecture is shown in Figure 2.4.

Workload Management Subsystem

The Workload Management System (WMS) comprises a set of Grid middleware components responsible for the distribution and management of tasks across Grid resources, in such a way that applications are conveniently, efficiently and effectively executed.

The core component of the Workload Management System is the Workload Manager (WM), whose purpose is to accept and satisfy requests for job management coming from its clients. For a computation job there are two main types of request: submission and cancellation. In particular the meaning of the submission request is to pass the responsibility of the job to the WM. The WM will then pass the job

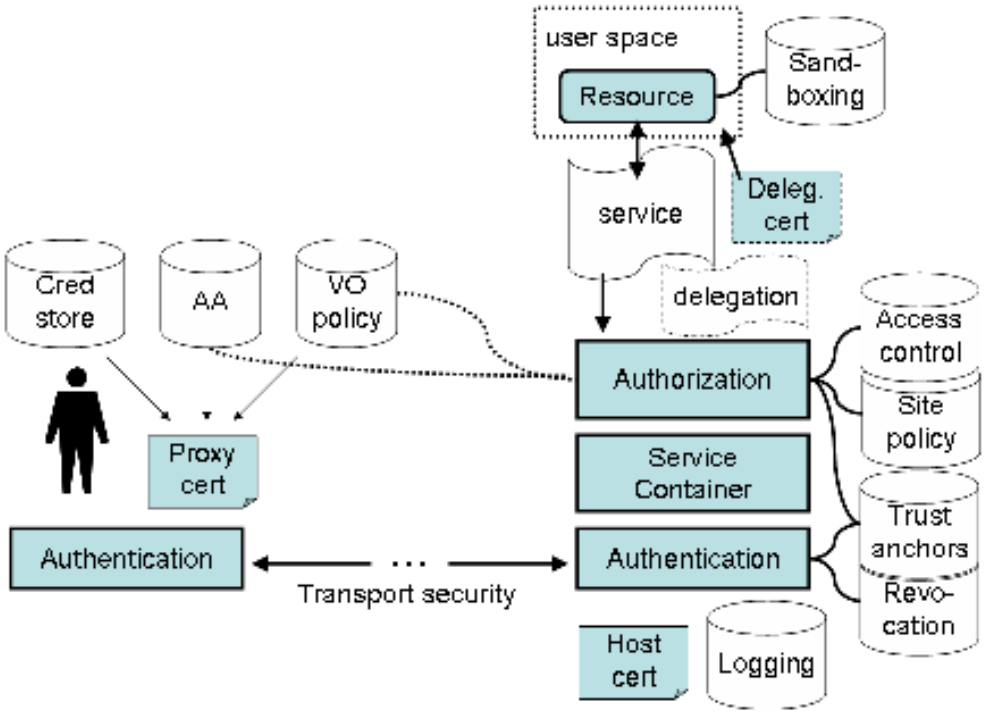


Figure 2.4: gLite security architecture.

to an appropriate CE for execution, taking into account the requirements and the preferences expressed in the job description. The decision of which resource should be used is the outcome of a matchmaking process between submission requests and available resources.

CHAPTER

3 Fire Dynamics Simulator

Fire Dynamics Simulator (FDS) is a computational fluid dynamics (CFD) model of fire-driven fluid flow. The software solves numerically a form of the Navier-Stokes equations appropriate for low-speed, thermally-driven flow with an emphasis on smoke and heat transport from fires.

3.1 Overview

Computational fluid dynamics models, in general, try to predict what will happen, quantitatively, when fluids flow, often with the complications of simultaneous flow of heat, mass transfer (e.g., perspiration, dissolution), phase change (e.g., melting, freezing, boiling), chemical reaction (e.g., combustion, rusting), mechanical movement (e.g., of pistons, fans, rudders), stresses in and displacement of immersed or surrounding solids [9].

Laws governing fluids are relatively simple and straightforward; motion, thermodynamics and some chemistry. Solutions, however, are rather complex. As a result, analytical methods are unusable in real-life problems. A common approach to reduce complexity is to replace the initial problem with a number of smaller, less-complex problems [31].

That is how FDS operates. Instead of solving equations for the whole volume surrounding a specific event, the user splits the flow domain into smaller sub-volumes. For FDS to work, the volume in question must be a rectangular area and subsequently the sub-volume grid must be rectilinear. A common simulation circle

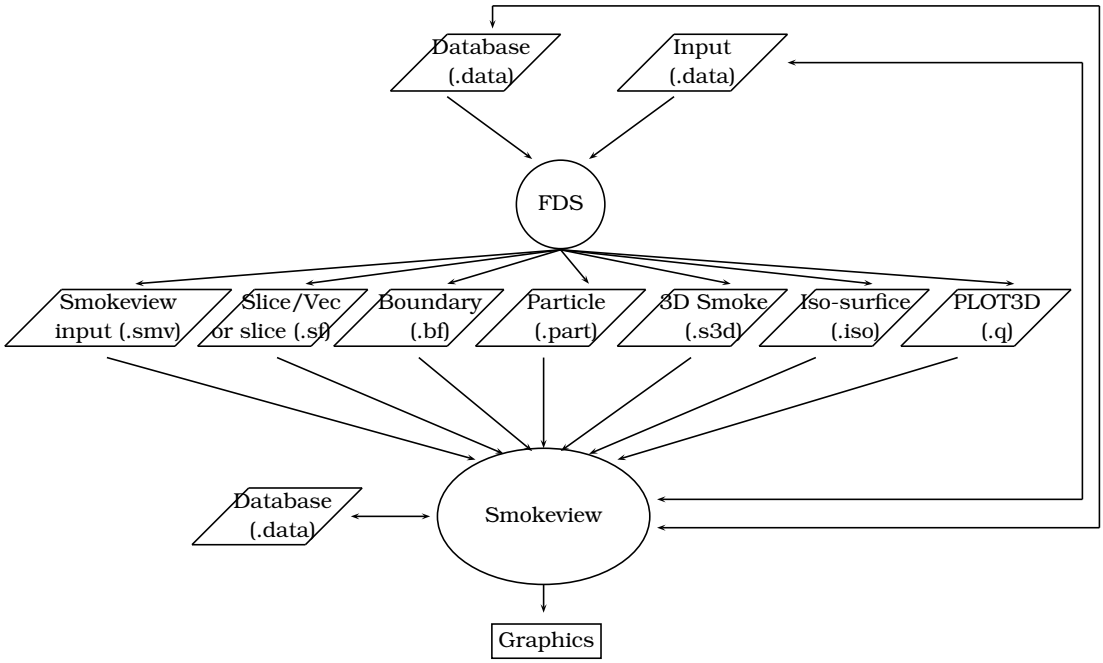


Figure 3.1: FDS data files and programs.

consists of three steps:

1. represent the objects in the flow domain and provide the "mesh" or "grid" (i.e., the splitting in sub-volumes procedure mentioned above).

2. run the simulation on the "mesh".

3. extract and visualize the data from the results produced from the simulation.

A diagram illustrating the data files and programs used in FDS is shown in Figure 3.1. From top to bottom, the diagram shows the user-provided input (Step 1), the simulation (Step 2), and the output files (Step 3). "Smokeview" is an application used to graphically represent the results of the simulation (see details below).

3.1.1 Step 1: The input file

Object representation and mesh generation is done with a user-provided input file. The input file is a text file containing all the essential information to describe the scenario under consideration. The most important inputs determine the physical size of the overall rectangular domain, the grid dimensions, and the additional geometrical features. Next, the fire and other boundary conditions must be specified. Finally, there are a number of parameters that customize the output files to capture the most important flow quantities [35].

A sample input file is shown in Listing 1. It describes two rooms (a large and a smaller one) connecting to each other with a door (See Figure 3.2). The scenario

states that a fire starts in the first (large) room and after a period of time, the door opens. The file `database.data` mentioned in `DATABASE='database4.data'` is an external file containing information about the properties of the object surfaces used (e.g., the material of which objects are made, details about how the objects react to specific fuels, etc.)

Listing 1 A sample input file of FDS.

```
&HEAD CHID='2subRm01z',TITLE='ATF Room Fire Test'  /
/
&GRID IBAR=48, JBAR=24, KBAR=24  /
&PDIM XBAR0=0.0,XBAR=4.7,YBAR0=0.0,YBAR=2.5,ZBAR0=0.0,ZBAR=2.5 /
/
&TIME DT=0.01,TWFIN=5.0 /
&MISC SURF_DEFAULT='SHEET METAL',
      DATABASE='database4.data',
      REACTION='POLYURETHANE' /

&SURF ID='burner',HRRPUA=1000. /

&OBST XB=2.60,3.20,0.95,1.55,0.0,0.10, SURF_IDS='burner',
                          'INERT', 'INERT' / burner
&OBST XB=3.65,3.75,0.0,0.87,0.0,2.5 / wall1
&OBST XB=3.65,3.75,0.87,1.63,2.0,2.5 / wall1
&OBST XB=3.65,3.75,0.87,1.63,0.0,2.0, T_REMOVE=2.5,
                          RGB=0.7,0.8,0.8 / door wall1
&OBST XB=3.65,3.75,1.63,2.5,0.0,2.5 / wall1

&VENT CB='XBAR' , SURF_ID='OPEN' / open right side of 2nd room

&PL3D DTSAM=5. / Plot 3D file every 5 secs
&PART DTPAR=0.5,NIP=100 /
```

3.1.2 Step 2: Simulation

Fires are generally known to be of the most complex phenomena to understand. The difficulties revolve about three issues [21]: fire scenarios are by default accidental. Thus, the number of possible cases to consider is gigantic. Secondly, the computational power needed to perform all the necessary calculations (bluff body aerodynamics, multi-phase flow, turbulent mixing and combustion, radiative transport, and conjugate heat transfer) is limited. Finally, what turns out to be the "fuel", in most cases, was never intended to be such. As a result, mathematical models describing the phenomena may not be available at all. After all, modeling

physical and chemical transformations of real materials as the burn is an area still under heavy development.

In order to continue, things must be simplified. Work begins with investigating a small number of scenarios that are easier to analyze and can be generalized over time instead of trying to develop a a methodology applicable to all fire problems. Moreover, like any other complex computational problem, descriptions of fire are idealized and approximations are be used widely.

Hydrodynamic Model In fluid dynamics, the Navier-Stokes equations, named after Claude-Louis Navier and George Gabriel Stokes are a set of nonlinear partial differential equations that describe the flow of fluids such as liquids and gases [16]. FDS solves numerically a form of the Navier-Stokes equations appropriate for lowspeed, thermally-driven flow with an emphasis on smoke and heat transport from fires. The core algorithm is an explicit predictor-corrector scheme, second order accurate in space and time.

Combustion Model For most applications, FDS uses a mixture fraction combustion model. The mixture fraction is a conserved scalar quantity that is defined as the fraction of gas at a given point in the flow field that originated as fuel. The model assumes that combustion is mixing-controlled, and that the reaction of fuel and oxygen is infinitely fast. The mass fractions of all of the major reactants and products can be derived from the mixture fraction by means of "state relations", empirical expressions arrived at by a combination of simplified analysis and measurement.

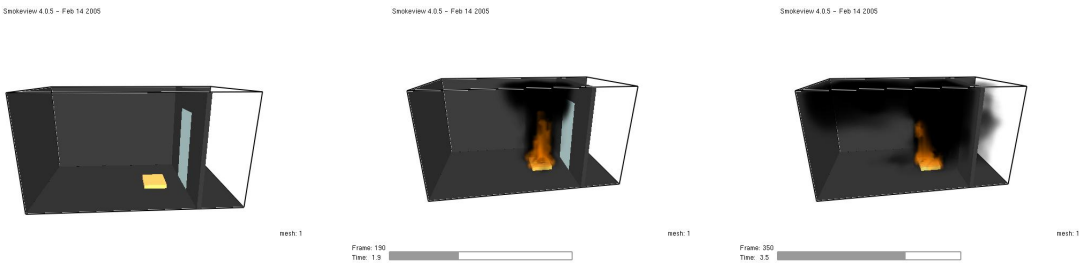
Radiation Transport Radiative heat transfer is included in the model via the solution of the radiation transport equation for a non-scattering gray gas, and in some limited cases using a wide band model.

Boundary Conditions All solid surfaces are assigned thermal boundary conditions, plus information about the burning behavior of the material. Heat and mass transfer to and from solid surfaces is usually handled with empirical correlations, although it is possible to compute directly the heat and mass transfer.

FDS can also simulate the presence of smoke detectors and sprinkles (when stated in the input file).

3.1.3 Step 3: Results

After the simulation is finished, results are written in a variety of output files (see Figure 3.1). Smokeview is a program written in C and Fortran 90, designed to read and visualize (using the OpenGL graphics library) both static and dynamic data from these files. Dynamic data is visualized by animating particle flow (showing location and "values" of tracer particles), 2D contour slices (both within the domain and on solid surfaces) and 3D level surfaces. 2D contour slices can also be drawn with colored



(a) Area overview before simulation

(b) Fire and smoke before the door opens

(c) Fire and smoke after the door has opened

Figure 3.2: The two room example.

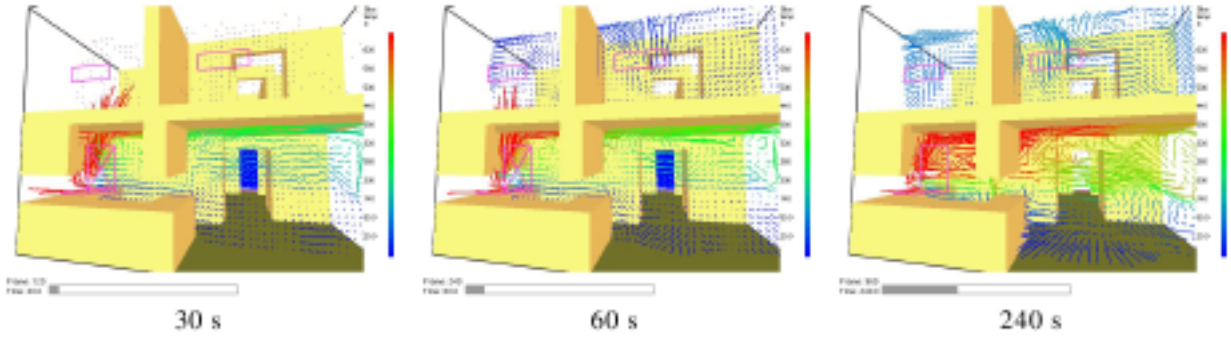


Figure 3.3: Dynamic data visualization.

vectors that use velocity data to show flow direction, speed and value. Static data is visualized similarly by drawing 2D contours, vector plots and 3D level surfaces [24].

In simple words, the user loads the output files into Smokeview, adjusts the scene as desired, selects what to watch (i.e., temperature on a specific slice of the mesh, smoke, water, etc.) and the entire simulation is presented to him as an animated sequence of frames.

Examples of Smokeview runs are presented. Figure 3.2 shows the result of the input file noted in Section 3.1.1. Figure 3.3 depicts snapshots of shaded vector plots in a vertical plane at different times after ignition. FDS is also capable of simulating large fires. Figure 3.4 shows a snapshot of a large oil fire simulation. The domain is 384m by 384m by 288m. Each tank is 84 m in diameter and 27 m tall. The grid is uniform with 6 m grid cells.

3.2 FDS Variations

FDS comes in two variations. The first is meant to be run on a single machine unlike the second which allows the distribution of sub-meshes (and the subsequent sub-

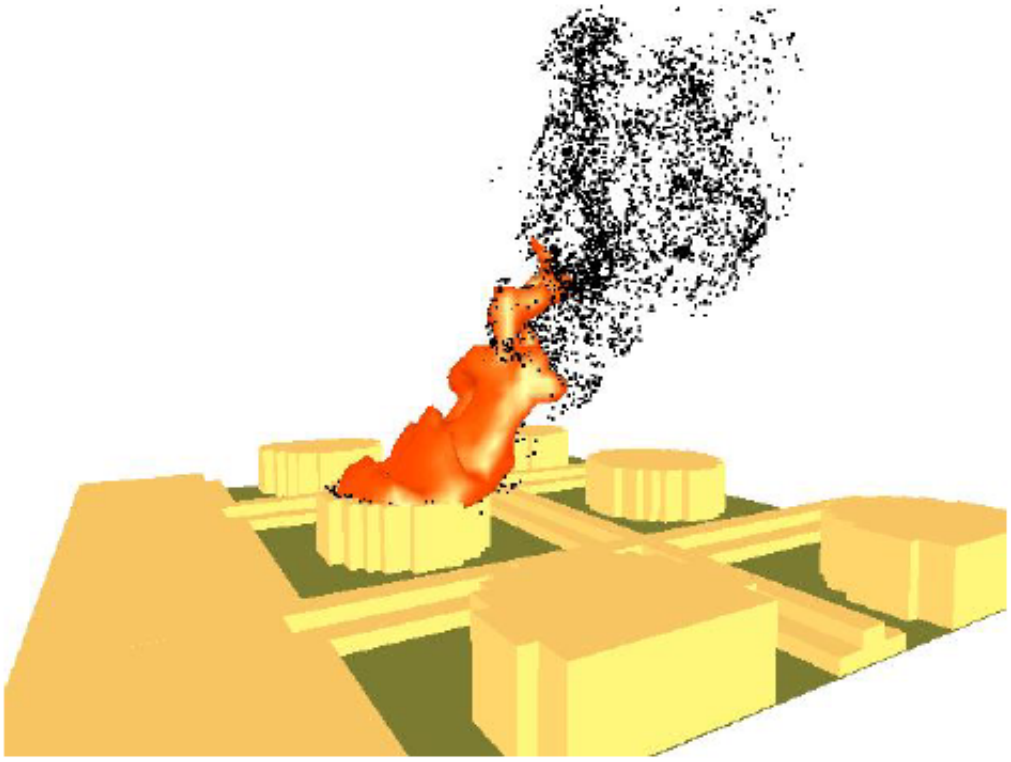


Figure 3.4: Large fire simulation.

computations) over a network (local or the internet), offering a boost in simulation times (see Chapter 4 for experiments and results). Of course, the computational core (i.e., the methods and algorithms used for the simulation) remain the same in both cases.

In order to run a calculation on more than one computer, FDS uses the Message-Passing Interface (MPI). MPI is a *specification* of a portable, practical, efficient and flexible interface (currently defined for C and Fortran languages) providing widely used standards for writing message passing programs [11].

The initial FDS job must be broken up into multiple meshes. The general idea is to assign a single mesh to each processor and run all simulations in parallel. Neighbour meshes, however, must exchange information considering the simulation itself. Also, all the results must be gathered by a single processor in the end to produce the output files. All this communication is handled by the MPI library.

The third version of FDS, (made for project DDEMA [2]) agents to scatter the meshes/computations over the internet. The agentised version of FDS is actually a wrapper over the MPI version, so again the philosophy is a mesh per agent. It's architecture is depicted in 3.5[1]. For the implementation of this version, JADE platform is used (see Section 2.1.2).

The main components of this version are:

MpiAgent.java (Java agent) This class provides several major functions such as detect and track neighbor agents, create MpiAdaptor object (see below) and start up its thread, handle `Send data` requests from the adaptor by means of wrapping the data into an ACL message and send it to the destination given by the adaptor, handle `Receive data` requests from the adaptor by storing the request information (source, tag, number of bytes, etc.) and waking up the adaptor when the data is received, handle asynchronous `Receive data` requests from the adaptor by storing the request in a queue and forward the data to legacy code when the data is received and finally handle `SYNC` requests from the adaptor by sending the `SYNC` to all neighbor agents and waking up the adaptor when it receives the `SYNC` from the others.

MpiAdaptor.java (MPI adaptor) This class acts as a thread host for the legacy code. Also relays `Send data` and asynchronous `Receive data` requests form MPI utility to the agent. Moreover, it forwards `Receive data` and `SYNC` requests from MPI utility to the agent (and waits until it is notified by the latter that it is OK to continue).

MpiWrapper.c (MPI wrapper) This is C code containing the two following JNI methods:

- **`Java_MpiAdaptor_fds_1execute()`** When MpiAdaptor is started by MpiAgent, first it creates the thread entity, then it calls this method to start legacy process.

[1] From "Fire Dynamics Simulator, Agent version: Design and User guide Documentation", by Haiping Zhang, (personal communication)

- **`Java_MpiAdaptor_mpi_1ireceiveDataArrived()`** This method is used to copy the data received from *asynchronous* requests to the legacy code buffers.

mpi_util.c (MPI utility) This file implements all MPI method calls that are used by the legacy code. Hence, this is the first step to pass data from low level FDS up to the agent level.

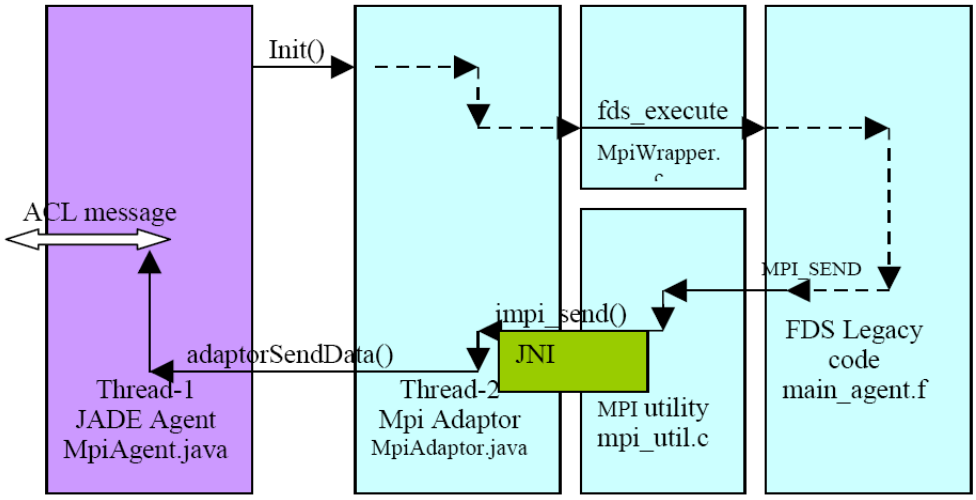


Figure 3.5: Example dataflow in agentised version of FDS.

Java programs can easily call C routines using JNI. However, the original code is written in Fortran. Thus, there is an extra layer between `MpiAdaptor` and the Fortran code written in C which plays the role of an interpreter between the Java and the Fortran code. In other words, there is a Java/JADE wrapper around the C-code which in return is a wrapper around FDS (Figure 3.6).

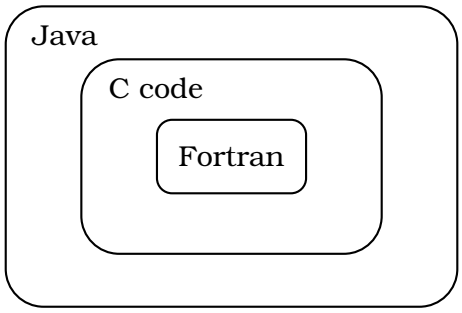


Figure 3.6: Code wrappers.

CHAPTER

4

Experiments and Results

The scope of the experiments was to run the MPI and agent versions of FDS, do performance measurements and determine the validity of the results produced by the agent version.

4.1 Machinery nostrum

4.1.1 Cyclone Cluster

Experiments involving MPI were run on *Cyclone*. *Cyclone* is one of the two Beowulf clusters [1] in the Department of Computer & Communication Engineering, University of Thessaly.

The setup of the machines is that of a typical cluster. All eleven computing nodes are connected to a local network through a switch (see Figure 4.1). There is also a master node with two network interfaces (one for the local network, one for the external network/Internet).

The master node is the entry point of the cluster, where the users are able to submit their job to a queue. Job scheduling is done by the master node as well, which is in charge of determining which node has spare clock cycles and send MPI "sub-jobs" to it. All nodes (including the master one) are identical in terms of internal components. Each one is equipped with a 2.6GHz, Pentium 4 processor, 512MB of RAM and an 80GB hard disk, resulting to a total of approx. 6GB of RAM and 960GB of disk space. The nodes are communicating to each other through a 100MBit switch.

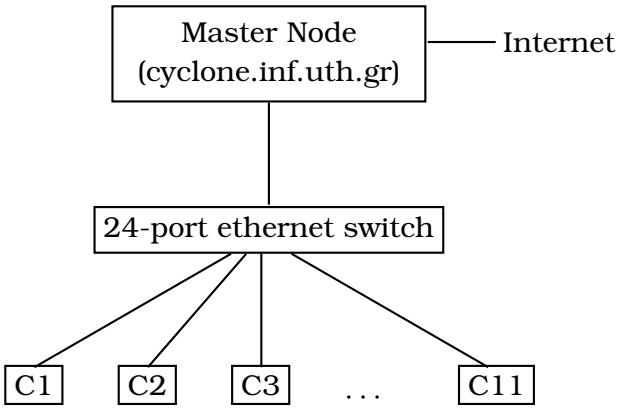


Figure 4.1: Cyclone network infrastructure.

Finally, current status of the cluster (online nodes, etc.) and real-time statistics (workload, memory usage, disk usage, etc.) are available through a web interface which can be found at `http://cyclone.inf.uth.gr`.

4.1.2 Local grid

In order to test the agent version of FDS, three computers were used. The master node of *Cyclone* (referred to as "Computing Node 1"), a 2.44 GHz – Pentium 4 machine with 512 RAM and a 80G SCSI disk ("Computing Node 2") and a 1.8 GHz – Pentium 4 with 256 RAM and a 80G IDE disk ("Computing node 3"). All of the nodes are connected to each other over a local network of 100MBit bandwidth.

When the simulation required more than three nodes, we run two agents on the same machine. The simulation scheme regarding Computing Nodes (CN) vs number of meshes (zones) is as following:

1 zone	:	CN 1
2 zones	:	CN 1, CN 2
3 zones	:	CN 1, CN 2, CN 3
4 zones	:	CN 1, CN 1, CN 2, CN 3
5 zones	:	CN 1, CN 1, CN 2, CN 2, CN 3
6 zones	:	CN 1, CN 1, CN 2, CN 2, CN 3, CN 3

The order of CNs in each row above is the same as the identification number of the corresponding mesh. For example, in the 5 zone case, meshes 1 and 2 are on CN1, meshes 3 and 4 on CN2 and mesh 5 on CN3.

4.2 Experiments

For all the experiments, the basic input file used is the one presented in Section 3.1.1, Listing 1. As mentioned before, this file defines two rooms separated by a door and simulates a burning fire for 5 seconds.

Six variations of this file were used, variations in terms of the different number of meshes used for the computation. In the first case, the entire space is assigned to a single mesh. In the second case, two meshes are defined, separating the area into two zones (see Listing 2). Same strategy is adopted respectively for the other cases where the same area is divided into three, four, five and six sub-meshes.

Listing 2 Definition of two non-overlapping sub-meshes.

```
/ No overlapping Zones
/ Zone 01
&GRID IBAR=24, JBAR=24, KBAR=24  /
&PDIM XBAR0=0.0,XBAR=2.4,YBAR0=0.0,YBAR=2.5,ZBAR0=0.0,ZBAR=2.5 /
/
/ Zone 02
&GRID IBAR=24, JBAR=24, KBAR=24  /
&PDIM XBAR0=2.4,XBAR=4.7,YBAR0=0.0,YBAR=2.5,ZBAR0=0.0,ZBAR=2.5 /
```

4.2.1 Performance measurements

We measure performance by means of measuring the time it takes for the simulation to complete.

Generally speaking, the procedure used to extract timings from the results was the following:

- run all versions (serial, parallel-MPI, parallel-agent) many times at different times and days to achieve statistically distributed workloads on the computers used,
- parse the overall summary (see Listing 3) stored in one of the output files called {jobname}.out and get the time consumed for the MAIN procedure (this is what we refer to as the representative time for the entire simulation),
- gather all different results for every simulation and for every submesh and calculate the average time for all three runs,
- calculate the maximum average time for the different submeshes,
- calculate the sum of average times for the different submeshes.

Listing 3 Extract of the overall simulation summary.

```
CPU Time Usage, Mesh   1

          CPU s      %
-----------------------
MAIN       215.01     100.00
DIVG        10.53       4.90
MASS        14.64       6.81
VELO        18.22       8.47
PRES         6.38       2.97
PART         0.00       0.00
DUMP        19.36       9.00
SPRK         4.89       2.28
RADI        28.62      13.31
COMB         2.78       1.29
COMM         3.17       1.48
```

Zones	Iterations	CPU time	Sum
1	505	178.45	178.45
2	502	183.77	183.77
3	502	182.31	182.31
4	502	181.24	181.24
5	502	171.58	171.58
6	502	189.52	189.52

Table 4.1: Timings for the serial version.

Results are summarized in several tables and figures. Tables 4.1, 4.2 and 4.3 show the results for serial[1], MPI and agent[2] version respectively. The first column indicates the number of zones (i.e., meshes) to which the space is separated. Second column contains information about the total number of iterations (timesteps) of the algorithm in order to reach the end of the simulation (5 seconds for this example). Third column contains the average actual timings for each mesh separated by commas.

Regarding the overall maximum mesh-times per file, we expect that, with the serial version, they will remain almost the same. When one processor is involved, there is no significant difference between computing one big mesh and two – or more – sub-meshes.

On the other hand, with the MPI version we expect that the more we split the initial area to smaller spaces, the more the computational times will decrease. This

[1] serial version reports the overall time, that's why there are not different times for different meshes

[2] agent version doses not support running a simulation on a single mesh. It needs two or more meshes in order to start

Zones	Iterations	CPU time	Sum
1	505	182.33	182.33
2	503	93.99, 99.61	193.60
3	502	92.16, 74.03, 29.78	195.97
4	503	92.53, 74.28, 16.29, 16.18	199.27
5	502	44.64, 44.88, 74.53, 16.54, 16.07	196.66
6	502	51.26, 51.70, 41.32, 39.91, 18.24, 18.04	220.47

Table 4.2: Timings for the MPI version.

Zones	Iterations	CPU time	Sum
2	503	98.91, 126.22	225.13
3	502	101.82, 106.80, 409.06	617.68
4	503	103.63, 100.39, 22.73, 655.85	882.59
5	502	50.82, 51.82, 100.30, 25.36, 1062.28	1290.58
6	502	58.47, 59.28, 58.55, 57.32, 717.35, 700.26	1651.24

Table 4.3: Timings for the agent version.

seems to be true. As shown in Table 4.2, since computations for different meshes are now performed simultaneously and the wall clock time for the execution is the maximum time over all the meshes (underlined numbers). Communication cost appears, which seems not to be very significant as one compares the total execution times (computation + communication) in the fourth column (sum) of the table. This is expected since all communication is done internally within the cluster nodes using the MPI functions.

Finally, we see that timings will increase for the agent version when more sub-meshes are used. The number of computations are the same as before but now it is important which computing node is used for each mesh. One can see that the longer times are always on CN 3. Communication cost here contains both communication between agents/meshes and the data passing from legacy code to agent level.

The above conclusions are confirmed in Figure 4.2, which shows the maximum average times for each version in a graphical way. The serial version is executed on all three computing nodes and the maximum (over the meshes) time is depicted with '□' for Cyclone, '♢' for CN 2 and '▽' for CN 3. The results are similar and show the computing power of the nodes.

The MPI version is executed on Cyclone and the maximum time decreases as the size of the bigger mesh decreases ('△' on Figure 4.2).

The agent version is used ('◃' on all CNs, '▹' on Cyclone) as discussed in Section 4.1.2. It is clear how the low power of CN 3 affects the maximum time, while in the case where all agents run on Cyclone, maximum time behaves as before.

Regarding the overall sum of times per case, we expect, for the serial version ('□' on cyclone, '♢' on CN 2 and '▽' on CN 3), that measurements are, once again, the same, whether multiple meshes are used or not.

For the MPI version ('△' on Cyclone), the results are expected to be very close

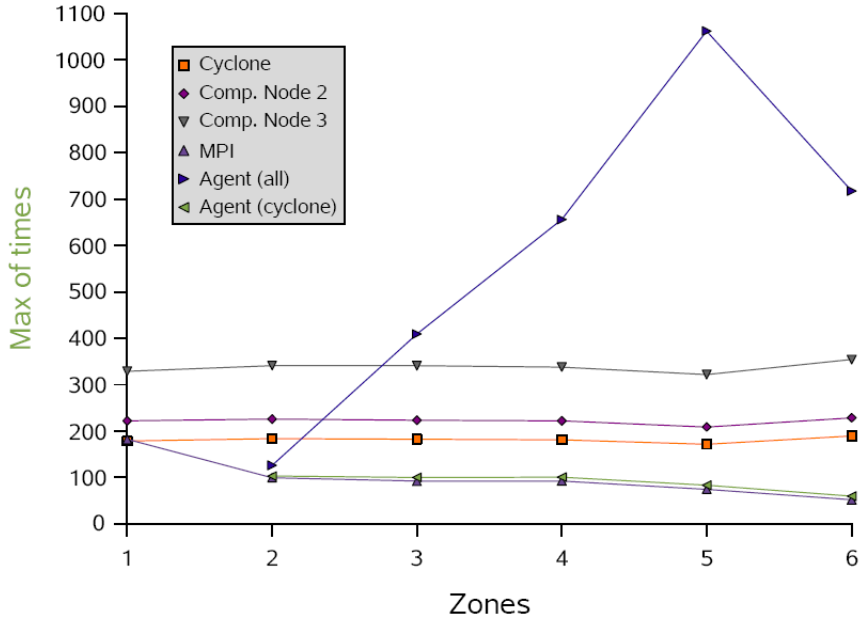


Figure 4.2: Summary of maximum times.

to the serial ones. Since all the nodes are identical, the sum of times needed for each mesh should be the same as the time needed for the entire space plus a small communication overhead. In other words, if we plot the results, we expect the MPI line to be just above the serial one.

For the agent version ('◃' on all CNs, '▹' on Cyclone only), however, we are stating again that communication between the machines is important. Moreover, we need even more clock cycles because of the chain of wrappers (data are passed from FORTRAN to C and finally to Java code - see Section 3.2). This is the reason the agent line is above the rest, when CN 3 is used and why the total times are increasing as the number of sub-meshes increases (see Figure 4.3).

4.2.2 Validation of results

We are interested in validating the results of the agent version, so that we can make sure that the wrapping is done correctly and the correct information is passed between the agents.

In order to do the validation we use *fds2ascii* [4], a FORTRAN utility which converts the results stored after the simulation in binary format (stored by FDS for time and memory usage optimization) to ascii files. These files contain the value of certain variables (as specified in the input file) at a two-dimensional slice of the space.

Having converted all output data for serial, mpi and agent simulations, we compare the results. In order to do that, we use a small command-line program written in Java which parses two fds2ascii output files and calculates the relative difference between

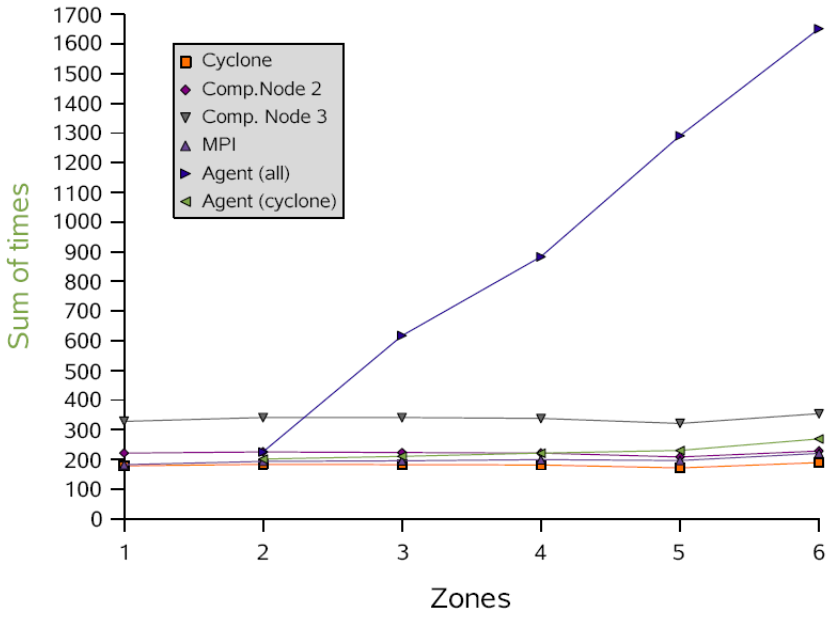


Figure 4.3: Summary of sum of times.

the values in terms of

$$\frac{||f-g||_2}{||f||_2} = \sqrt{\frac{\int\int [f(x,z)-g(x,z)]^2\, dxdz}{\int\int f(x,z)^2 dxdz}}$$

where f contains the values from the first file and g the values form the second file. Integration is calculated numerically using the trapezoidal rule

$$\int_a^b f(x)dx \simeq \frac{h}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + \ldots + 2f(x_{N-2}) + 2f(x_{N-1}) + f(x_N)\right)$$

We use relative difference (i.e., percentage) in order to have more meaningful results. We also use the L2 norms instead of maximum of differences because the scheme that FDS uses, needs random values for specific quantities at the initial step. As a consequence, we are interested in the difference over all the area.

The results of the comparison are shown in tables 4.4, 4.5 and 4.6. The quantities used for the comparison are `TEMPERATURE, HRRPUV and MIXTURE_FRACTION`.

As expected, most of the results are close to zero. The difference between the serial and the agent version behaves as the difference between the serial and the MPI one (which is tested by NIST), while there is no difference between the MPI and the agent version. This ensures the validity of the latter.

Serial - MPI				
Zones	Mesh	TEMPERATURE	HRRPUV	MIXTURE_FRACTION
2	1	0.109	0.000	0.261
	2	0.161	0.250	0.255
3	1	0.112	0.000	0.256
	2	0.129	0.200	0.138
	3	0.080	0.000	0.239
4	1	0.198	0.000	0.456
	2	0.251	0.296	0.268
	3	0.041	0.000	0.373
	4	0.092	0.000	0.223
5	1	0.032	0.000	1.284
	2	0.133	0.000	0.357
	3	0.160	0.310	0.222
	4	0.056	0.000	0.208
	5	0.058	0.000	0.168
6	1	0.138	0.000	4.359
	2	0.226	0.000	0.634
	3	0.303	0.317	0.308
	4	0.312	0.797	0.398
	5	0.064	0.000	0.355
	6	0.155	0.000	0.373

Table 4.4: Percentage difference between Serial and MPI version.

Serial - Agent				
Zones	Mesh	TEMPERATURE	HRRPUV	MIXTURE_FRACTION
2	1	0.109	0.000	0.261
	2	0.161	0.250	0.255
3	1	0.112	0.000	0.256
	2	0.129	0.200	0.138
	3	0.080	0.000	0.239
4	1	0.107	0.000	0.254
	2	0.137	0.149	0.126
	3	0.028	0.000	0.193
	4	0.081	0.000	0.183
5	1	0.060	0.000	1.995
	2	0.046	0.000	0.132
	3	0.099	0.234	0.140
	4	0.085	0.000	0.421
	5	0.170	0.000	0.579
6	1	0.136	0.000	4.281
	2	0.224	0.000	0.626
	3	0.295	0.296	0.288
	4	0.306	0.742	0.390
	5	0.066	0.000	0.354
	6	0.144	0.000	0.348

Table 4.5: Percentage difference between Serial and Agent version.

MPI - Agent				
Zones	Mesh	TEMPERATURE	HRRPUV	MIXTURE_FRACTION
2	1	0.000	0.000	0.000
	2	0.000	0.000	0.000
3	1	0.000	0.000	0.000
	2	0.000	0.000	0.000
	3	0.000	0.000	0.000
4	1	0.152	0.000	0.424
	2	0.182	0.222	0.197
	3	0.045	0.000	0.284
	4	0.058	0.000	0.097
5	1	0.044	0.000	0.622
	2	0.117	0.000	0.227
	3	0.120	0.172	0.190
	4	0.126	0.000	0.744
	5	0.148	0.000	0.489
6	1	0.003	0.000	0.040
	2	0.006	0.000	0.016
	3	0.020	0.041	0.040
	4	0.015	0.734	0.024
	5	0.014	0.000	0.055
	6	0.012	0.000	0.030

Table 4.6: Percentage difference between MPI and Agent version.

Bibliography

[1] (2005, June) Beowulf project overview. [Online]. Available: http://www.beowulf.org/overview/index.html

[2] (2005, June) Ddema project. [Online]. Available: http://www.cs.purdue.edu/DDEMA

[3] (2005, June) Distributed european infrastructure for supercomputing applications - deisa. [Online]. Available: http://www.deisa.org/

[4] (2005, June) Fds/smokeview tools. [Online]. Available: http://fire.nist.gov/fds/refs/tools.html

[5] (2005, June) The fipa website. [Online]. Available: http://www.fipa.org/

[6] (2005, June) Geant website. [Online]. Available: http://www.geant.net/

[7] (2005, June) General parallel file system. [Online]. Available: http://www-1.ibm.com/servers/eserver/clusters/software/gpfs.html

[8] (2005, June) Global grid forum. [Online]. Available: http://www.gridforum.org/

[9] (2005, June) Introduction to computational fluid dynamics. [Online]. Available: http://www.cham.co.uk/website/new/cfdintro.htm

[10] (2005, June) Kaariboga mobile agents. [Online]. Available: http://www.projectory.de/kaariboga/index.html

[11] (2005, June) Message passing interface (mpi). [Online]. Available: http://www.llnl.gov/computing/tutorials/mpi/

[12] (2005, June) Moore's law, the future - technology and research at intel. [Online]. Available: http://www.intel.com/technology/silicon/mooreslaw/

[13] (2005, June) Proactive website. [Online]. Available: http://www-sop.inria.fr/oasis/ProActive/

[14] (2005, June) Seti@home. [Online]. Available: http://setiathome.ssl.berkeley.edu/

[15] (2005, June) Voyager java development platform. [Online]. Available: http://www.recursionsw.com/voyager.htm

[16] (2005, June) Wikipedia, the free encyclopedia – navier-stokes equations. [Online]. Available: http://en.wikipedia.org/wiki/Navier-Stokes_equations

[17] H. C. Allan Snavely, Greg Chun, "Benchmarks for grid computing," Apr. 30 2003. [Online]. Available: http://www.sdsc.edu/~allans/sigmetricsfinal.pdf

[18] M. Baker, R. Buyya, and D. Laforenza, "Grids and grid technologies for wide-area distributed computing," *SOFTWARE -PRACTICE AND EXPERIENCE*, vol. 32, no. 15, pp. 1437–1466, 2002.

[19] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "Jade - a white paper," Sept. 2003. [Online]. Available: http://exp.telecomitalialab.com/upload/articoli/V03N03Art01.pdf

[20] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," Aug. 31 1992.

[21] F. R. D. Building and F. Fire Research Laboratory, "Fire dynamics simulator (version 4): Technical reference guide," Nov. 2003. [Online]. Available: http://fire.nist.gov/bfrlpubs/fire04/PDF/f04100.pdf

[22] J. Cao, "Self-organizing agents for grid load balancing," *Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pp. 388–395. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/GRID.2004.57

[23] J. Cao, D. P. Spooner, J. D. Turner, S. A. Jarvis, D. J. Kerbyson, S. Saini, and G. R. Nudd, "Agent-based resource management for grid computing," *Scientific Programming*, vol. 5, 2002. [Online]. Available: http://www.dcs.warwick.ac.uk/research/hpsg/documents/CaoJ.ARMGC.pdf

[24] G. P. Forney and K. B. McGrattan, "User's guide for smokeview version 4: A tool for visualizing fire dynamics simulation data," Aug. 2004. [Online]. Available: http://fire.nist.gov/bfrlpubs/fire04/PDF/f04098.pdf

[25] I. Foster, "What is the grid? a three point checklist," *GRIDToday*, July 20 2002.

[26] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International J. Supercomputer Applications*, vol. 15, no. 3, 2001.

[27] I. Foster, N. R. Jennings, and C. Kesselman, "Brain meets brawn: Why grid and agents need each other," *3rd International Conference on Autonomous Agents and Multi Agent Systems, New York USA*, July 22 2004.

[28] I. Foster and C. Kesselman, *The Grid: Blueprint for a new computing infrastructure*. Los Altos, CA 94022, USA: Morgan Kaufmann Publishers, 2003.

[29] G. Fox, D. Gannon, and M. Thomas, *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, March 2003.

[30] M. Fukuda, Y. Tanaka, N. Suzuki, L. F. Bic, and S. Kobayashi, "A mobile-agent-based pc grid," *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), June 25th, 2003,Seattle, WA*.

[31] D. Hunt. (2001, Dec.) Computational fluid dynamics. [Online]. Available: http://www.tessella.com/literature/supplements/PDF/cfd.pdf

[32] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng, "Overview of the ptolemy project," July 2 2003. [Online]. Available: http://ptolemy.eecs.berkeley.edu/publications/papers/03/overview/overview03.pdf

[33] M. Luck, P. McBurney, and C. Preist, *Agent Technology: Enabling Next Generation Computing*. Univ.Southampton, Jan. 2003. [Online]. Available: http://www.agentlink.org/admin/docs/2003/2003-48.pdf

[34] E. Mangina, "Review of software products for multi-agent systems," June 2002. [Online]. Available: http://www.agentlink.org/admin/docs/2002/MAS.pdf

[35] K. McGrattan and G. Forney, "Fire dynamics simulator (version 4): User's guide," Sept. 2004. [Online]. Available: http://fire.nist.gov/bfrlpubs/fire04/PDF/f04099.pdf

[36] S. A. Neuendorffer, "Actor oriented metaprogramming," 2004. [Online]. Available: http://ptolemy.eecs.berkeley.edu/publications/papers/04/StevesThesis/twoside.pdf

[37] J. R. Pugh, "Actors - the stage is set," *ACM SIGPLAN Notices*, vol. 19, no. 3, pp. 61–65, 1984.

[38] J. Schopf, "Grids: The top ten questions," *Scientific Programming*, vol. 10, no. 2, 2002.

[39] M. Wooldridge, *Multiagent Systems*. John Wiley & Sons, Ltd, 2002.