The Design, Usage, and Performance of GridUFO: A Grid based Unified Framework for Optimization

Asim Munawar [a,*,1], Mohamed Wahib [a], Masaharu Munetomo [b], Kiyoshi Akama [b]

[a] *Grad. School of Information Science & Tech., Hokkaido University, Sapporo, Japan*
[b] *Information Initiative Center, Hokkaido University, Sapporo, Japan*

Abstract

We present GridUFO (Grid based Unified Framework for Optimization), a Service Oriented Architecture (SOA) compliant Problem Solving Environment (PSE) that allows the user to implement/share metaheuristics based optimization algorithms over a Grid. GridUFO eradicates the shortcomings of earlier projects and provides a unified approach for using algorithm/problem pair over a Grid in the easiest possible "plug & play" manner. This framework allows the users to concentrate on the actual application development, by hiding all the complexities involved in a Grid, without compromising on the functionality and flexibility promised by a Grid. This paper provides a detailed overview of the GridUFO infrastructure, specifically the way it deals with optimization algoirthms and objective functions, handles Service Level Agreements (SLAs), and follows SOA. We also present various results achieved, that demonstrate both the utility and performance of GridUFO under various application workloads and scenarios.

Key words: Grid computing, service oriented architecture, problem solving environment, metaheuristics based optimization algorithms, optimization framework

* Corresponding Author
Email addresses: asim@ist.cims.hokudai.ac.jp (Asim Munawar), wahibium@ist.cims.hokudai.ac.jp (Mohamed Wahib), munetomo@iic.hokudai.ac.jp (Masaharu Munetomo), akama@iic.hokudai.ac.jp (Kiyoshi Akama).
[1] Present Address: Division of Large Scale Computing Systems, Information Initiative Center, Hokkaido University, North 11, West 5, Sapporo 060-0811, JAPAN
phone: +81(11)706–3759 facsimile: +81(11)706–3759

1 Introduction

Distributed computing and its close relative, Grid computing, are becoming less rare with each passing day. Grid computing allows a secure and coordinated sharing of globally distributed resources spanning several physical organizations[13]. Service Oriented Architectures (SOAs) underlie several of the current Grid initiatives and reflect the current Grid computing infrastructures, where the participants offer and request application services. SOA defines standard interfaces and protocols that enables the developers to encapsulate resources of different complexities and values as services that clients can access without the knowledge of their internal workings[11]. Embracement of SOA by Grid computing is one of the key reasons for the success of Grids[14]. The motivation for the development of the Grid was to offer the computing as a service to the users without them knowing the acutal location of the physical resources. Therefore, we argue that in true spirits of a Grid the framework should support SOA. Moreover, Service Level Agreement or SLA should be offered to define the level of service the user can expect.

Every field of applied sciences and engineering, rely on the solution to optimization problems in one way or the other. The solvers for such problems include a vast set of algorithms, ranging from gradient descent to algorithms that mimic nature like metaheuristics based algorithms. As stated by No Free Lunch (NFL) theorem [36], we cannot have a single optimization algorithm that gives us good results for all the problems in the entire problem space, and the average performance of all the algorithms over the entire problem space is the same. This forces us to develop a framework incorporating different specialized algorithms from which the user can select the algorithm of his own choice, to solve the optimization problem at hand. Metaheuristics based algorithms have a tremendous potential for parallelization over modern parallel computing paradigms like Grid. Therefore, we will restrict our discussion only to metaheuristics algorithms in this paper. The high rate of increase in distributed resources, data deluge, multicore systems, and high speed networks makes the parallel and distributed implementation of algorithms more vital than ever before.

In this paper we present a Grid based Unified Framework for Optimization (GridUFO), a SOA compliant Problem Solving Environment (PSE) that allows the user to implement and share metaheuristics based algorithms over a Grid. The main target of this research is to harness the power of the Grid, to maximize the efficiency of metaheurisitcs based optimization algorithms, while keeping all the low-level details hidden from the application developer. To achieve this target, GridUFO uses open standards and protocols to hide the complexities of the Grid from the user without compromising the flexibility and functionality offered by the Grid. Word "Unified" in GridUFO have several implications: (1) GridUFO unifies the features of other similar projects at one place (not to mention the extra functionalities provided by the framework), (2) It provides a single platform to solve any kind of

metaheuristics algorithm, (3) It unifies the Grid technologies with the optimization algorithms to take the optimization frameworks concept to a new level.

Owing to the very gradual learning curve of Grid technologies, researchers and application developers are reluctant to use Grid computing models. Application developers usually waste a lot of time and energy in understanding the basics of Grids, Grid middlewares and Grid configurations. Moreover, due to the lack of mature tools, this process is very sensitive and error prone. To ease this process by providing an abstraction to the user is the basic motivation behind this paper. GridUFO can significantly reduce the development time and allow the developer to concentrate on the actual problem, i.e. optimization in the case of GridUFO. We believe that computing models like Grid computing and utility computing (utility computing is a kind of Grid computing where the users only pay for the services/resources they use) is the future of distributed computing, and projects like GridUFO can encourage the application developers to shift to Grid technologies. GridUFO provides a smooth transition from conventional parallel models to modern distributed computing technologies.

GridUFO is a step towards Virtual Innovative Laboratory (VIL), proposed by Munetomo[28] (2006). VIL seeks for realizing virtual laboratory that innovates automatically to find optimal solutions or designs, by combining robust evolutionary search and simulator program of the target problems. VIL intends to replace a part of human designer's trial-and-error process.

Most of the similar optimization frameworks attempted in the past (see Sect. 2) either lacks the required flexibility or the Grid compatibility, asserting them practically unusable for the modern distributed computing paradigms. They include Grid computing, utility computing and the more recent, cloud computing paradigms. Moreover, the Grid based counterparts of GridUFO do not provide the functionality in the true spirits of a Grid and closely resembles conventional parallel architectures. GridUFO on the other hand is a true SOA compatible Grid based design and offers all its functionalities as services to the users. The contract of the service is determined by the Service Level Agreement (SLA) protocol. SLA protocol used in GridUFO is defined in Sect. 3.5. From a user's point of view GridUFO is a framework that offers services to different classes of users namely: (1) Users who want to share an optimization algorithm with other GridUFO users, (2) Users who want to share an optimization problem with other GridUFO users, (3) Users who want to solve an optimization problem using an optimization algorithm already registered with the framework. There are two Service Access Points (SAPs) of GridUFO namely, (1) Web Services Resource Framework (WSRF) based Web service that can be consumed by any client application independent of the operating system and architecture of the system, (2) Fully integrated 2nd generation Web portal (with custom portlets) that allows the user to access all the services through any Web browser. Some other salient features of GridUFO are listed in Sect. 2.

The novelty of GridUFO lies in its unique way of treating the optimization algorithms and the objective functions. In a usual setting both these entities are tightly connected and inseparable. However, in the case of GridUFO these two entities are loosely coupled and can be developed by different developers. Moreover, both the entities are Quality of Service (QoS) conscious and formally declare the services provided by them using the SLD file (SLD file is a Service Level Description file written by using a proposed markup language discussed in Sect. 4). User is allowed to use any pair of optimization algorithm and objective function. The QoS between an optimization algorithm and an objective function is negotiated during the SLA process by the SLA service of GridUFO (see Sect. 3.5). GridUFO enforces the algorithm developer to use an API that we call "GridUFO API". An algorithm written using this API is automatically recognized by the system on registration. All the calls to the objective function are made through this API and are hence hidden from the user. Similarly an objective function with a proper IDL file is also recognized by the system automatically. Using the tools including GridUFO API, SLD, and IDL GridUFO allows a very loose and flexible connection between optimization algorithm and objective functions. It is important to note that this approach of GridUFO is very different from the normal workflow approach. Workflow allows us to arrange execution of different pieces of codes in a sequence; on the other hand GridUFO's approach allows a continuous interaction between two independent pieces of codes running at remote locations. GridUFO make use of GridRPC[33], GridMPI[20], and GridUFO's job scheduler for distributed implementation over the Grid (see Sect. 3.6). We also propose MetaHeuristics Markup Language (MHML), an XML based language that acts as an interface between the user and the framework.

This paper is organized as follows: In the next section we will discuss the related work done in this area. In Sect. 3 we will discuss the design and implementation of the system in detail. Section 4 explains the proposed markup language. Section 6 gives us some theoretical analysis and empirical results for the overheads involved in Grids. We conclude the paper in Sect. 7 with some guidelines for possible improvements in the system.

2 Comparison with the Past Work

Several optimization frameworks have been developed for solving different optimization problems in the recent years. However, we will only discuss the most noticeable projects, e.g., NEOS [9,16,10], Folding@Home[21], Nimrod/O[3], GEODISE[8], OSP[1], and GE_HPGA[23]. Most of these projects are either non Grid compliant or lack the features promised by a real Grid environment. Moreover, most of these projects do not work in a black box manner (important for metaheuristics algorithms), do not allow the user to add new algorithms, and almost all of the projects completely ignore the service oriented aspect of the Grid. Therefore, we argue, that

Table 1
Comparison of GridUFO with the work done in the past.

	Scope	Black-box Optimization	Architecture	Middleware	Information Exchange	Adding New Algorithm by User	Security	Algorithm Parallelization	User Interface	User Guidance to the Algorithms
NEOS	Any Optimization	Not Supported	Trivial Web Application	Non-Standard	Plain Sockets	Allowed (Through Management)	No	Predefined	Web-based Job Submission	Text Describing Each Algorithm
Folding@Home	GROMACS or Protein Structure Optimization	Not Supported	Distributed	SMP Client	Client-Server Sockets	Not Allowed	Yes (2048 bit Digital Signature)	Independent Work Units	–NA–	–NA–
Nimrod/O	Non-linear Optimization	Not Supported	Event Driver	OGSA Compliant	Active Sheets	Not Allowed	Yes (GSI)	Predefined	Web Portal	Ontology
GEODISE	Fluid Dynamics	Not Supported	ASP	OGSA Compliant	Environment API	Not Allowed	Yes (GSI)	Predefined	Matlab Tool box/ Portal	Ontology
OSP	Decision Support System	Not Supported	ASP	Aggregated Components	AMPL/MPL	Not Allowed	Yes	Predefined	Web Portal	Designated Tools
GE-HPGA	Global Optimization with an Island Model GA	Supported	Trivial Distributed Application	OGSA Compliant	Environment API	Not Allowed	Yes (GSI)	Fixed Island Model (problem divided to independent sup-problems)	None	None
GridUFO	Global Optimization with different metaheurisits	Supported	SOA	OGSA Compliant	MHML (XML Based)	Allowed (Through Portal)	Yes (GSI)	Algo Writer defines his own Parallel/Distributed Model	Web Portal & Web Service	SLD + SLA

projects (frameworks) attempted in the past are not in agreement with the true spirit of Grids as most of them ignores the service oriented aspect of the Grid. The term Grid Oriented Genetic Algorithms (GOGAs) were first introduced by Imade et al. [18] in 2003. Since then, some research has been done on the GOGAs [23,19,17], which is still far from mature. We argue that presenting a new Grid based algorithm is not so meaningful unless the user has a robust Grid based framework to use that algorithm (as setting up a Grid is not a trivial task). At the same time, a Grid based framework is not so meaningful unless it offers the algorithms through Grid services and allows the user to add new algorithms. A comparison of some of the most noticeable projects with GridUFO is given in table 1.

SOA although ignored by most of the earlier projects is important for building a flexible framework. Keeping this in consideration GridUFO is SOA compatible and all the features of GridUFO are available as standard Grid services. Moreover, GridUFO is compatible with Open Grid Services Architecture (OGSA) compliant Grid middlewares and allows the user to solve global optimization problems over a Grid in a black box fashion. Some salient features of GridUFO as compared to the previous work can be listed as follows:

- GridUFO is a SOA compliant Grid based architecture.
- Most of the projects done until today requires manual deployment of the application on all the resources before starting the execution but GridUFO provides a mechanism for automatic deployment.
- GridUFO has well defined interfaces between the user and the framework (based on MHML). MHML is not limited to GridUFO and can be used with any other similar framework.
- GridUFO supports black box optimization.
- GridUFO allows the algorithm and objective function to be developed by different developers.
- GridUFO allows the user's to add and share new metaheuristics based algorithms.
- GridUFO allows the algorithm developer, objective function developer, client and the resource owner to maintain their own set of policies. SLA service of GridUFO uses these policies to finalize a contract between these entities. GridUFO uses a multi-stage SLA to finalize a QoS contract.

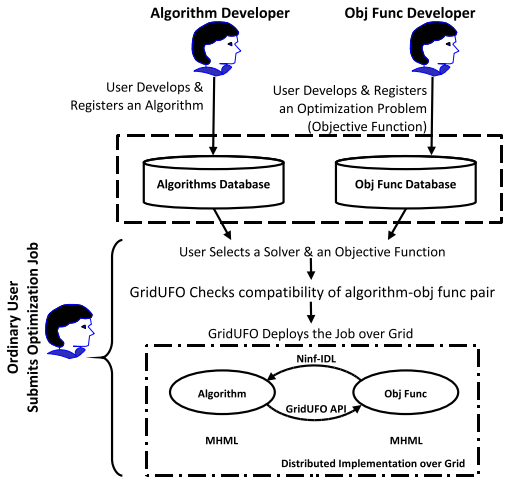

Fig. 1. GridUFO: The core concept.

- GridUFO has two SAPs namely; GridPortal and/or Grid based Web Service. It offers all the services through either of the SAPs.
- GridUFO emphasizes simplest possible interfaces for the users without compromising on the flexibility and functionality offered by the Grid. This can be considered as the biggest challenge in design and implementation of GridUFO.

In the coming sections we will explain all the above mentioned features in detail. In short, GridUFO addresses above mentioned shortcomings of the work done in the past and provides solutions to these problems in consent with the true spirit of Grids. Moreover, GridUFO provides extra features that were not present in any of its past counterparts.

3 The GridUFO Infrastructure

In this section we will discuss the design & implementation of the proposed framework. We will start from the core concept and then move towards the key technologies involved in realizing this concept.

3.1 The Core Concept of GridUFO

The core concept of GridUFO at an abstract level is shown in Fig. 1. The figure depicts the unique way in which GridUFO handles the optimization algorithm and objective function (two main modules of an optimization job). In a standard implementation of algorithms, these two modules are tightly linked with each other and are inseparable. However, GridUFO is unique in the way it treats these two modules. GridUFO maintains separate databases for the algorithms and for the objec-

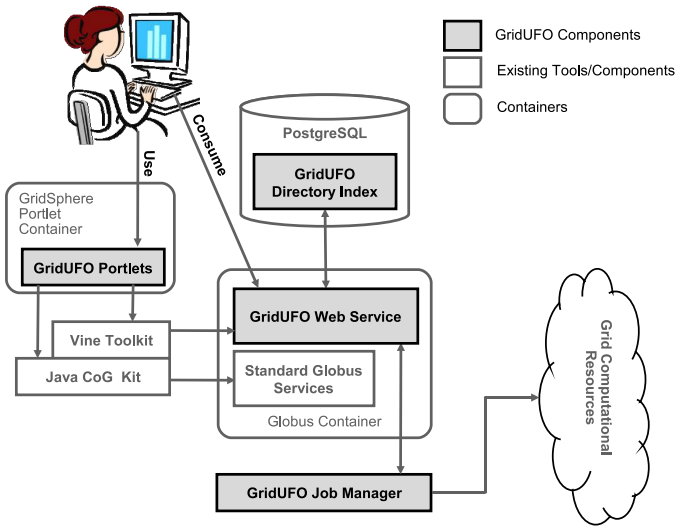

Fig. 2. Abstract level architecture of GridUFO (shows the most notable components of the system).

tive functions as shown in the figure. The optimization algorithm and the objective function can potentially be developed by different developers and get registered with the framework. The client (user) is allowed to use any algorithm/objective function pair. The contract of service between these two entities is defined by the SLA service of GridUFO. GridUFO automatically executes both the algorithm and the objective function in a distributed manner over the available resources. This unified approach towards all kinds of metaheuristics based algorithms allows the user to test an algorithm against many different objective functions; similarly, user can try to solve an optimization problem by using different algorithms available in the framework.

3.2 Architecture of GridUFO

Figure 2 shows the abstract level architecture of GridUFO. A brief description of the main components of the framework is as follows:

(1) *GridUFO's Web Service:* is a WSRF compliant web service running inside a Globus Toolkit's[12] Web service container. This Web service can be consumed by any client application, independent of the architecture and the platform that application is using. Web Services Description Language (WSDL) file declares the interface of the GridUFO's Web service to the client application. GridUFO's Web service offers different services to the user including SLA service, and job submission service.
(2) *GridUFO's Web Portal:* is a fully integrated 2nd generation Gridsphere[31] portal with JSR 168 compliant GridUFO portlets installed. Each portlet corresponds to one or more services, provided by the GridUFO's Web service. The

portlets can be thought of as client applications consuming the GridUFO's Web service on user's behalf.

(3) *GridUFO's Directory Index:* is a PostgreSQL based database containing all the algorithms and the objective functions registered with the framework. It also maintains logs/history of the jobs submitted to the framework.

(4) *GridUFO's Job Manager:* is responsible for maintaining a job waiting queue, and running the jobs over the available computational resources. It uses Ninf-G's [2] InvokeServer[35] functionality and/or GridUFO's Scheduler to schedule the submitted job on the available resources.

(5) *GridUFO's Resources:* refers to the computational resources only. For the time being GridUFO's resources can include any resource using Globus Toolkit WS GRAM, Globus Toolkit Pre-WS GRAM, Condor[15], SSH, NAREGISS [25], or UNICORE [6]. The user can access other kinds of Grid compatible resources (e.g. remote data source or equipment using GridFTP) by employing the GridUFO API.

Another important component is GridUFO's scheduler. It is a simple job scheduler which can be replaced by any other well known scheduler or super scheduler without any major modification in the system.

3.3 The Software Stack

The GridUFO builds on the existing Grid technologies and tools for performing "data intensive" computing on distributed resources. It adds new tools for solving optimization problems as Grid services. There are numerous applications of such tools in both academia & industry. Figure 3 shows the software stack used to realize the design of GridUFO. Bottom layer is "platform infrastructure layer". It comprises the Grid fabric including physical computational resources, networks, data storage centers etc. Second layer is "low-level middleware layer". Low-level middleware provides a secure and transparent access to resources. GridUFO uses tools like Globus Toolkit 4.0.x [12], scheduler in low-level middleware layer. Third layer is "upper-level middleware layer". The semantic gap between the "low-level middleware layer " and the "application layer " makes it difficult to build useful applications on top of low-level middleware directly. Therefore, "upper-level middleware layer" is introduced in between which is often called as "user-level middleware layer". Some of the important tools used by GridUFO in this layer include Java Cog Kit[22] (abstraction on top of Globus), Vine Toolkit[2] (a toolkit for using custom made Web services), Ninf-G [35], Condor [15] (scheduler) etc. "Upper-level middleware layer" provides tools for application development and the aggregation of distributed resources. The top most layer is "application layer". This layer can have various Grid based applications and/or PSEs that can make use

[2] Ninf-G is a reference implementation of GridRPC [33]

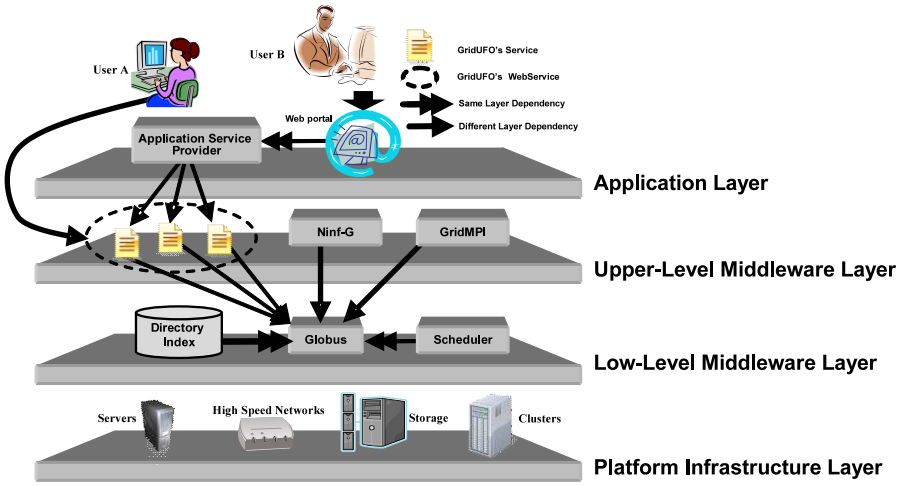

Fig. 3. GridUFO's layered structure.

of low-lying layers to solve a specific group of problems. GridSphere [31] (Grid portal) used by GridUFO sits at this layer.

It is clear from the figure that, GridUFO provides two Service Access Points (SAPs) to the user. It is deliberate to have GridUFO offering two different SAPs in the middleware and the application layers as shown in the figure. In Fig. 3, "User A" uses the Web portal to access the framework, while "User B" uses the Web service as SAP. Hence, GridUFO is understandable to both humans (via Portal), and machines (via Web Service). This feature of GridUFO closely resembles the future incarnation of Semantic Grid [3]. Theoretically speaking, GridUFO's Web portal is just a client service consuming the GridUFO's Web service on behalf of the user.

3.4 SOA Compliance

We have used GridUFO's WSRF compliant Web service to implement SOA, hence enabling the users to employ platform independent standard Internet protocols to access the services offered by GridUFO. GridUFO's Web service provides the core services that orchestrate all the components in the system. The four basic services offered by the framework are:

(1) *Register Service:* allows the developer to register a new algorithm or an objective function with the framework. The user is required to submit the SLD file along with the code (only of C language) of the algorithm and objective function respectively. It is important to note that the register service is only available through the Web portal and not through the Web service directly.

[3] Semantic Grid is an extension of the current Grid in which information and services are given well-defined meaning, understandable by both humans and machines, better enabling computers and people to work in cooperation

(2) *Retrieve Service:* when queried, returns a list of all the algorithms and objective functions registered with the framework along with their SLDs.
(3) *SLA Service:* is a service of GridUFO that finalizes the service contract between four QoS-aware entities namely; algorithm, objective function, client, and resources owners.
(4) *Job Submit Service:* allows the user to submit an optimization job to the framework through a job submission file. Job submission file contains the name of the algorithm, name of the objective function, and configuration of the job [26]. The user has an option to wait for the job to finish, or get the results emailed whenever the job is finished.

Note that both SLD and the job submission file are MHML files. As the name suggests, SLD describes the specification about an algorithm or an objective function. See Munawar et al. (2007)[26] for details on SLD.

3.5 SLA in GridUFO

Service Level Agreement (SLA) is that part of a service contract where the level of service is formally defined. SLA is mostly used to refer to the contracted delivery time (of the service) or performance. Usually SLA is between two parties the client and the application or resources. However, in the case of GridUFO there are four QoS-aware components each having its own policy, namely the algorithm, the objective function, the client submitting optimization job and the resource owners. Hence, GridUFO uses a multi-stage SLA concept to finalize a contract that does not violate any policies and is acceptable to the client.

Four service policies are involved in the final contract. They are defined by SLD files of the Optimization algorithm, SLD file of the Objective Function, the Job submission file, and the resource policy file. As mentioned earlier, in case of GridUFO

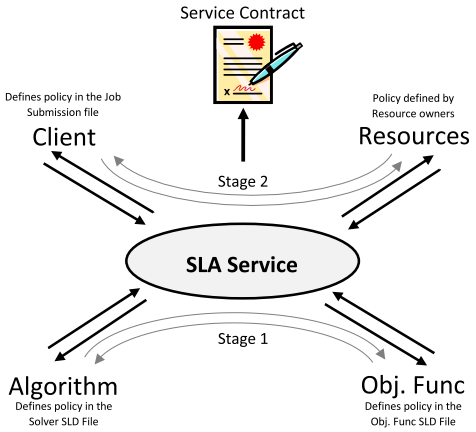

Fig. 4. SLA Rectangle of GridUFO. Note: All the policies are defined in MHML format.

Table 2
SLA metrics for stage 1.

Algorithm	Obj Func	Description
Name	Name	Used for selection and maintain logs.
Algorithm Class	Compatible Algorithm Classes	Algorithm SLD defines the class of the algorithm. Obj Func SLD must define the compatible classes of algorithms. Class means if an algorithm is SimpleGA, Simulated Annealing, EDA etc.
Type	Type	Is it real, integer, binary, permutation etc.
Parallel or Not	Parallel or Not	Does it use GridMPI for parallelization or not. Used in Stage 2 of SLA.

the algorithm and the objective functions can be written by different developers and the user has an option to select any optimization algorithm/objective function pair at the time of job submission. So, it is important to check compatibility of the algorithm and the objective function before the job submission. Therefore the SLA service of GridUFO is slightly more complicated than the usual, as it defines the contract between four entities (as shown in Fig. 4). In this section we will discuss the SLA strategy used by GridUFO. SLA is one of the most important features of GridUFO not present in any of its predecessors.

The SLA is performed in two steps. Step 1 occurs at the application layer while step 2 occurs at the middelware layer. In the first stage SLA is negotiated between the algorithm and the objective function. If this stage fails, an error is returned to the user without proceeding to the second stage. This is due to the fact that incompatible algorithm/objective function pair will result in meaningless results. Table 2 shows the SLA metrics used for deciding the compatibility of an algorithm with an objective function during the stage 1 of SLA. This performance metrics only tell us if the algorithm/objective function pair is compatible or not.

If the algorithm and objective function are compatible with each other the SLA service moves on to the stage 2 of SLA. In stage 2 the service checks the available resources, their policies and the requirements of the applications. User defines the policy in the job submission file. User can either opt to minimize the resource use, to minimize the execution time or to minimize the cost (applicable for utility computing). User can also set limits to the minimum number of resources used, maximum number of resources used, maximum amount of time, maximum cost etc. The framework estimates the service on the basis of available resources and resource use policy and returns an error if the required resources are not available. After that user can either change his preferences or wait till the resources get free. If stage 2 is also passed the contract if finalized and job is forwarded to the job manager.

3.6 Distributed Implementation (GridUFO's Approach)

As compared to conventional parallel computing paradigms, Grid is very different in its approach towards distributed implementation of applications/algorithms. GridUFO employs GridMPI, GridRPC and GrifUFO's job scheduler for distributing the job over the available computational resources. All these tools depend on the low-level Grid middleware (Globus in the case of GridUFO) for their working in one way or the other. GridMPI is not a very good choice to be used in a Grid environment where the scheduling takes place dynamically. Moreover, GridMPI is not fault tolerant and have a severe restriction of globally unique IP's. Therefore, we have restricted ourselves to the use of GridMPI within the cluster only and GridRPC for inter cluster executions. This use of GridRPC and GridMPI for parallelization is borrowed from Takemiya et al. (2006)[34] and is shown in Fig. 5.

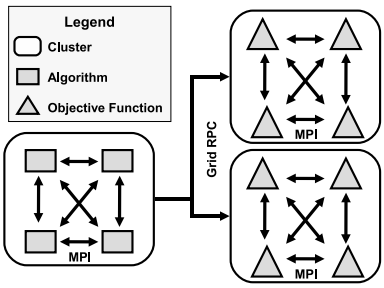

Fig. 5. Unique hybrid GridRPC + GridMPI approach to parallelization.

3.6.1 Automatic Deployment

Two major problems in the area of modern distributed computing are debugging and deployment of applications. Heterogeneity, of Grids makes automatic deployment very difficult. Therefore, most of the projects attempted in the past use manual deployment of applications, i.e., the application must be deployed on all the servers before the execution [34]. Manual deployment is in extensive use but it is not in consent with the true concept of Grids. Even though, GridUFO does not provide any feature for the debugging part, it provides a very elegant solution to solve the problem of deployment. It provides a mechanism for automatic deployment of applications over heterogeneous resources without any user intervention.

GridUFO uses a unique approach for implementation of algorithms over distributed environment. Whenever GridUFO receives a new job it reads the following MHML files:

(1) Job submission file (written by the user submitting the job). Name of the algorithm and objective function are given in this file.
(2) Algorithm SLD (written by algorithm developer).
(3) Objective Function SLD (written by objective function developer).

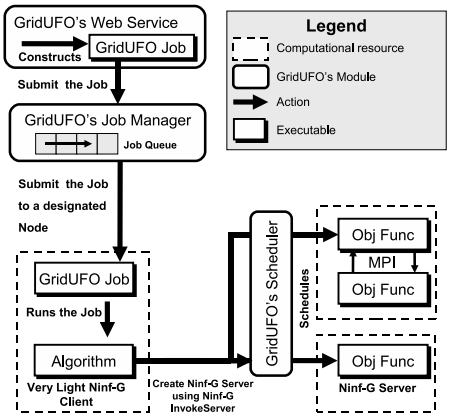

Fig. 6. Deployment over GridUFO.

(4) Algorithm Configurations (written by user submitting the job), can be left blank if user wants the default configurations.
(5) Objective Function Configurations (written by user submitting the job), can be left blank if user wants the default configurations.

For deployment purposes GridUFO uses the information given in the SLD files for the algorithm and the objective function respectively. We give two options to the user: (1) provide executables for different kind of environments, (2) provide the code with compilation command on different environments. If the user selects the second option in the SLD file, GridUFO automatically compiles the code on the resource (depending on the environment and SLD) before execution. If user selects the first option GridUFO deploys the appropriate executable depending on the environment.

The job manager of GridUFO is responsible to deploy the application code before running it. GridUFO strategy for deployment is shown in Fig. 6. A very light Ninf-G client is initiated on a node. The Ninf-G client then instantiates the Ninf-G server by using the GridUFO scheduler. This is done by using the *InvokeServer* functionality of Ninf-G. This method has some overheads but, it is very reliable, flexible, and robust. It provides an automatic mechanism to recover from errors using the check pointing mechanism provided by the scheduler.

3.6.2 GridUFO API

An existing algorithm can be converted into a GridUFO compatible algorithm by using a simple API (GridUFO API) provided to the algorithm developer. Most important functions provided by this API are shown in Fig. 7. The API can be used to make calls to the objective function; it also allows reading and parsing the MHML based configuration files. Calls to the objective function must be made through the API. User is also required to call the "init" and "destroy" function of the API at the start and end of the algorithm respectively.

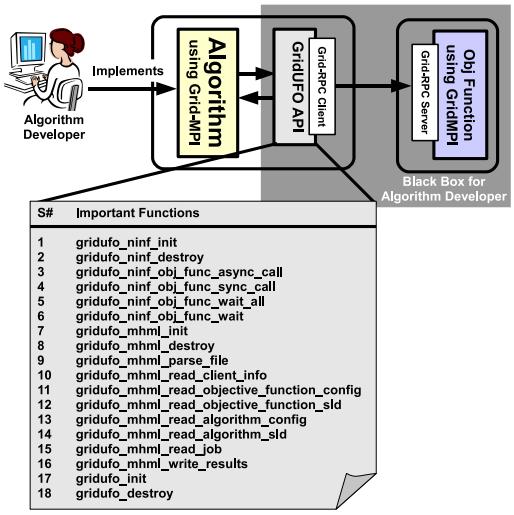

Fig. 7. GridUFO API.

Objective function is a C code which is treated as a black box by the system. User is required to write a code for the objective function along with the Ninf-IDL[35] interface file.

After implementing a GridUFO compatible algorithm or objective function the user can submit it to the framework for registration using "Register Service"of the framework. User must also submit the SLD files for the algorithm and the objective function respectively.

3.7 User's Perspective

From the user's perspective GridUFO is a PSE meant for solving optimization problems over a Grid. There are four distinct types of GridUFO users:

(1) *Administrators:* can access the framework only through the Web portal (and not through the Web service). Administrator manages the user accounts, certificates etc. Administrator can monitor all the jobs running in the framework and interrupt them in the middle.
(2) *Algorithm Developers:* write new algorithm for the framework using the GridUFO API. He also writes the SLD for the algorithm before submitting it to the framework. Algorithm developer uses the Register Service offered by the framework. This service is only accessible through the Web portal.
(3) *Objective Function Developers:* write GridUFO compatible objective function and add it to the framework along with the Ninf-IDL and the SLD file. Objective function developer also uses the Register Service offered by the framework to register the objective function. This service is only accessible through the Web portal.
(4) *Ordinary users:* are the users who want to execute an optimization job over

the framework. Most of the GridUFO users will be of this type. They can use the framework through the Web portal or by consuming the GridUFO's Web service directly. Figure 8 depicts the sequence of operations in the framework for submission of a new job. Before submitting a job the user can acquire a list of all the available algorithms and the objective functions by using the "Retrieve Service", after that user can select any algorithm and objective function and request a guidance from the "SLA Service", after that the job is submitted through the "Job Submit Service". This service calls the SLA service again to finalize the contract of service and then forwards the job to the job manager.

The user needs to register with the framework to acquire a valid user certificate before starting to use the services. This can be done by using the Web portal.

3.8 GridUFO's Web portal

GridUFO's Web portal is one of the two SAPs available to the user. It provides many other services beside the service to submit an optimization job. Main services offered through the portal are: (1) Register a new user, (2) Administer the jobs, (3) Register/Remove an algorithm, (4) Register/Remove an objective function, (5) Retrieve a list of available algorithms and objective functions, (6) Check optimization algorithm-objective function compatibility, and (7) Submit optimization job. GridUFO's Web service only provides services number 5,6, and 7; all other tasks should be done through the Web portal directly.

We have designed JSR 168 compliant custom portlets for each of the service provided by the portal. These portlets runs inside a GridSphere portlet container. The portal allows the user to access GridUFO through any Web browser without in-

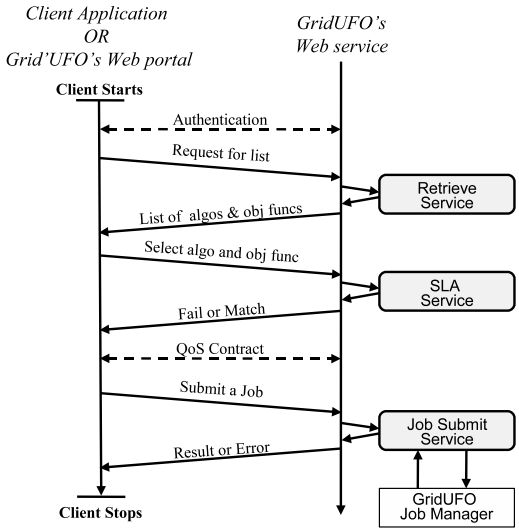

Fig. 8. Job submission scenario from user's perspective.

stalling any extra client application. The screen shot of the GridUFO portal (job submission portlet) is shown in Fig. 9.

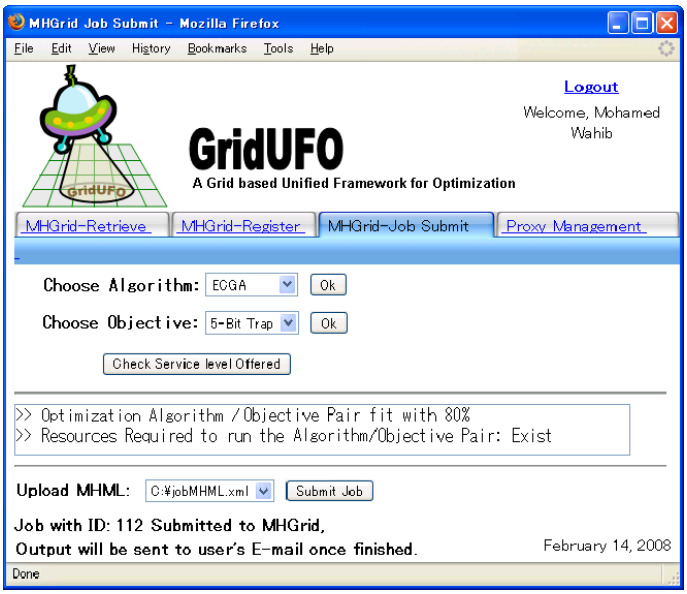

Fig. 9. Screenshot of GridUFO Job Submission portlet. (Screenshot has been resized and slightly modified to make it more appropriate for printing)

3.9 Algorithms for GridUFO

Theoretically speaking, a serial or a conventionally written parallel algorithm can run on GridUFO without any modifications. However, there is a need for algorithms exclusively designed for the Grid environments. Grid computing environment is very different from its conventional counterparts. Grid has a hierarchical structure that supports different Grain sizes, at different levels. Bottom layer consists of multicore resources like multicore CPUs, middle layer consists of medium grain resources like clusters, while the top layer is coarse grained layer[27]. Therefore, hierarchical algorithms like GE-HPGA[23] are more suitable for implementation over Grid. Grid based algorithms: (1) should support interconnections of loosely coupled distributed applications; (2) must be able to tolerate communication delays up to 100's of milliseconds; (3) must be fault tolerant; (4) should support late-binding [4] ; (5) should support dynamic migration and; (6) must rely on external data resources (both real-time data streams and archival data resources), whenever required. Due to the inherent nature of Metaheuristics algorithms if properly designed can easily fullfil all these requirements.

[4] In modern distributed computing the decision on a specific service instance is not made until needed, this is called late-binding

4 MHML

All the communication between the user and the framework is done by using a proposed language that we call MetaHeuristics Markup Language (MHML). Here, we give only a brief introduction to MHML, for a complete description see Munawar et al. (2007) [26]. Standardization of communication interfaces is often ignored by the developers of similar projects attempted in the past (as shown in table 1). A standard communication interface however, can lead to a flexible design. It provides interoperability, and reduces the chance of human error. GridUFO forces the developer to use a standard interface for the algorithm as well as the objective function. The interface can be standardized using different methods; however, XML appears to be the most promising language for such a purpose.

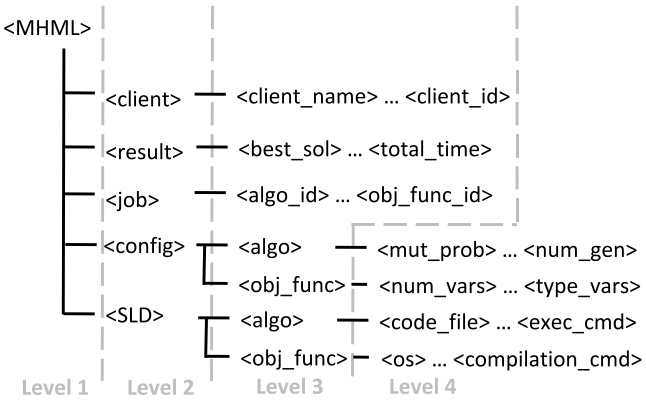

Fig. 10. Top level hierarchy of MHML with important tags.

MHML can be considered as an extension of the work presented in E. Alba et al. (2003) [5]. E. Alba et al. (2003) fail to address some very important issues regarding the configuration of an optimization algorithm. MHML provides many advantages over E. Alba et al. (2003) and can be applied to a greater number of cases. It provides advanced features like SLD, which is used to define the service level provided by the respective entity. MHML has the capability to represent:

(1) *Job Configuration* or job submission file is a file containing the configuration about the job submitted to the framework. Information in this file includes the name of the algorithm, name of the objective function, and execution preferences.
(2) *Algorithm or Objective Function SLD* is submitted during the registration of the algorithm or the objective function with the framework. SLD contains the algorithm or objective function related information. For example, the compatible OS, execution commands, interfaces, and other specifications.
(3) *Algorithm or Objective Function Configuration* are submitted along with the job configuration file. They contain information regarding the configurations for the algorithm and the objective function. For example a typical configuration file for a Genetic Algorithm (GA) may contain the population size, the

```
// GridUFO Initializations
gridufo_init(); // includes (gridufo_ninf_init(); gridufo_mhml_init();)
gridufo_mhml_parse_file(); // it will automatically parse the required mhml (SLD) files

// Algorithm Code Starts
Init(Population);
while termination_criteria not reached do
       for i=0 to PopulationSize/2 do
              [Parent_1, Parent_2] = Selection(Population);
              [Child_1 , Child_2 ] = Crossover(Parent_1, Parent_2);
               NextPopulation[2*i] = Mutation(Child_1);
           NextPopulation[2*i + 1] = Mutation(Child_2);
           gridufo_ninf_obj_func_async_call(NextPopulation[2*i]);
           gridufo_ninf_obj_func_async_call(NextPopulation[2*i + 1]);
       endfor
       gridufo_ninf_obj_func_wait_all();
       Population ← NextPopulation;
endwhile

// GridUFO Termination
gridufo_mhml_write_resutls(); // writes the results to an mhml file
gridufo_destroy(); // includes (gridufo_mhml_destroy(); gridufo_ninf_destroy();)
```

Fig. 11. GridUFO compatible simple Genetic Algorithm (sGA).

mutation probability etc. These files can be left blank if the user wants to use the default configurations.

(4) *Client Information* is a part of all the MHML files and it contains the information about creator of that MHML file. This is mainly used for logging.

(5) *Results/Errors* is the only MHML file generated by the framework. The results/errors are compiled into MHML format before submission to the user.

MHML is presented to the user as an XML schema [26]. A top-level hierarchy of MHML is shown in Fig. 10.

5 Use Case

GridUFO framework covers a wide spectrum of technologies and understanding the role of each component in the system can be difficult. Therefore, in this section we provide a step-by-step use case for running a simple Genetic Algorithm (sGA) over

```
/* This is a Ninf-IDL file*/
Module obj;
Define obj_func(IN int in_length_of_chromosome, IN int in_chromosome[length], OUT int *out_fitness)
"sga on rpc"
Required "obj_func.o"
{
       extern int obj_func(int length, int *x);
       *out_fitness = obj_func(in_length_of_chromosome, in_chromosome);
}
```

```
/* Implementation obj_func.c File */
int obj_func(int Len, int *Array)
{
       res ← 0;
       for i = 0 to Len do
              res ← res + Array[i];
       endfor

       return res;
}
```

Fig. 12. GridUFO compatible OneMax Objective Function and the required Ninf IDL file.

```
<?xml version="1.0"?>
<MHML  xmlns="http://www.mhgrid.hokudai.ac.jp/mhgrid"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.mhgrid.hokudai.ac.jp/mhgrid solve.xsd">
       <!-- Client Information -->
       <client>
              <client_ip>192.168.0.235</client_ip>
              <client_id>1</client_id>
       </client>
       <!-- Job Submission Information -->
       <job>
              <algo_id>3</algo_id>
              <objective_function_id>1</objective_function_id>
              <algo_config_file>NULL</algo_config_file>
              <objective_function_config_file>NULL</objective_function_config_file>
              <job_type>MinimizeExecutionTime</job_type>
       </job>
       <!-- Configuration Information -->
       <config>
              <algo>
                     <algo_id>1</algo_id>
                     <length>100</length>
                     <population_size>50</population_size>
                     <LS_prob>1.0</LS_prob>
                     <max_generation>300</max_generation>
              </algo>
              <objective_function>
                     <param></param>
              </objective_function>
       </config>
</MHML>
```

Fig. 13. Job Submission MHML file

GridUFO to solve one-max optimization problem. We will also discuss the registration process of the algorithm and the objective function with the framework. As a first step, we need to convert the algorithm to a GridUFO compatible algorithm. A C language implementation of sGA can easily be converted to GridUFO compatible algorithm by using the GridUFO API (Fig. 7) as shown in Fig. 11. It is clear from the figure that all the lower level details are kept hidden from the user. Similarly, an objective function can be converted to a GridUFO compatible objective function by providing a Ninf_IDL file as shown in Fig. 12. The user is than required to write the MHML files defining the SLD of the objective function and the algorithm. After this step the algorithm and the objective function must be added to the framework using the Register service of GridUFO. Now any user registered with the GridUFO framework can write a job submission MHML file and select the above mentioned algorithm and objective function and run the job on the Grid. This job submission MHML is shown in Fig. 13. The job submission file is then submitted to the GridUFO where the SLA takes place. In case the SLA passes the job is submitted to the scheduler for scheduling over the Grid. On the other hand in case of any error the error is returned back to the user.

The registration of the optimization algorithm and the objective function may seem to be a tedious job. However, the reality is that this is a one time job and once registered the algorithm and the objective function can be used by anyone by simply submitting a job submission MHML file.

6 Results & Discussion

Although the scope of Grid is very broad and is bound to expand in the near future to incorporate many more advantages over conventional parallel environments, yet distribution of an application over heterogeneous resources (to reduce execution time) will remain one of the main advantages/styles of using Grids. In this section we will give a theoretical analysis of the maximum achievable speedup for the framework. We will also give some empirical results that demonstrate both the utility and performance of GridUFO under various application workloads and scenarios.

6.1 Theoretical analysis of GridUFO

The speed-up S of a parallel algorithm over a parallel computing environment can be defined as T_s/T_p. Where T_s and T_p denote the execution time when the algorithm is executed in serial and parallel, respectively. It is not easy to formulate a single equation for GridUFO, as it allows the user to run any kind of metahuristic algorithm on a Grid. Therefore, we will give theoretical analysis of only one algorithm, i.e., parallel Linkage Identification using Nonlinearity Check algorithm (pLINC) by Munetomo et al. (2003) [30].

We assume that, $\lambda(n)$ is the total time taken by the algorithm, $\gamma(n)$ is the total time taken for fitness evaluations, and n is the problem size. For pLINC algorithm $\lambda(n)$ can be divided into three parts, serial part $\lambda_{serial}(n)$(time taken by selection and InterGA) and parallelizable parts $\lambda_{linkage}(n)$(time taken for evaluating linkages), $\lambda_{IntraGA}(n)$ (time taken by IntraGA step). We can define the speed-up as:

$$S(n,p) = \frac{\lambda_{serial}(n) + \lambda_{linkage} + \lambda_{IntraGA} + \gamma(n)}{\lambda_{serial}(n) + \frac{\lambda_{linkage}(n)+\lambda_{IntraGA}(n)+\gamma(n)}{p} + O(n,p)} \tag{1}$$

where p is the total number of computational nodes, and $O(n,p)$ is the parallelization overhead. According to Amdahl's law[7], equation for maximum speed-up $S^{max}(n,p)$ can be obtained by assuming $O(n,p) = 0$.

Now we will try to find the speed-up achieved by running the same algorithm on multiple clusters as compared to a single cluster implementation. In this case the single cluster implementation will become the T_s and the multiple cluster implementation will become T_p. For this purpose we will define ω as

$$\omega(n) = \lambda_{linkage}(n) + \lambda_{IntraGA}(n) + \gamma(n) \tag{2}$$

Now we can compute T_s and T_p as follows:

- *Single cluster implementation* is suitable when $\lambda(n) \approx \gamma(n)$ OR $\lambda(n) > \gamma(n)$. It is recommended that a single cluster implementation is used when the fitness evaluation is very light. We can compute T_s as:

$$T_s = \lambda_{serial} + m\left(\alpha\frac{\omega_c}{s} + O_{intra}\right) \tag{3}$$

  where s is the number of computational nodes in a cluster, α is the parallelism factor of the cluster, O_{intra} gives the parallelism overhead within the cluster, m is the total number of computational subgroups or total number of clusters in multiple cluster implementation, and ω_c is equal to ω/m.

- *Multiple cluster implementation* is suitable for the cases where $\gamma(n) \gg \lambda(n)$. For such cases GridUFO automatically runs the optimization algorithm on one cluster and distributes the fitness evaluations among other real or virtual clusters. We can ignore the time taken by the optimization algorithm (linkage identification, mutation, crossover, and selection) $\lambda(n)$ as it is much less than the total time used for fitness evaluation $\gamma(n)$ and Grid overheads $O(n, p)$, but we will keep it for comparison purposes with single cluster case. Therefore, using this information T_p can be computed as:

$$T_p = \sum_{i=1}^{m} O^i_{inter} + \lambda_{serial} + \left(\alpha\frac{\omega_c}{s} + O_{intra}\right) \tag{4}$$

  where O_{inter} is the communication overhead between clusters, m is the total number of clusters , s is the number of nodes in a single cluster, and α is the parallelism factor of a cluster (it is a function of problem size and CPU specifications).

In Eqs. 3 and 4, we can define:

$$u = \left(\alpha\frac{\omega_c}{s} + O_{intra}\right) \tag{5}$$

Therefore, and we will also neglect the term λ_{serial}, because usually it is too small compared to other values. We can now write the equation for maximum speed-up offered by GridUFO as:

$$S^{max} = \frac{T_s^{max}}{T_p^{min}} = \frac{mu^{max}}{mO_{inter}^{min} + u^{min}} \tag{6}$$

where T_s^{max} is the maximum total time to execute pLINC algorithm over a single cluster, T_p^{min} is the minimum total time to execute pLINC algorithm over m clusters, O_{inter}^{min} is the minimum inter cluster communication overhead, u^{max} is the maximum total time taken by slowest cluster to execute the $(1/m)^{th}$ part of the algorithm, and u^{min} is the minimum total time taken by the fastest cluster to execute

Table 3
Empirical results obtained by running a dummy problem over GridUFO using a Simple Real GA (time is shown in seconds).

T_f (Sec)	T_s (Sec)	T_p (Sec)	T_s / T_p	u^{avg} (Sec)	O_{inter} (Sec)
0.001	3	241	0.012	0.0667	16.62
0.01	14	228	0.0614	0.66	15.1
0.1	102	240	0.425	6.6	15.56
0.5	499	233	2.141	33	13.337
1	993	295	3.366	66	15.26
2	1984	366	5.42	132	15.6
5	4955	771	6.42	330	29.4
10	9903	1218	8.13	660	37.205

$(1/m)^{th}$ part of the algorithm. For further details on Eq. 6, see Munawar et al.[27] (2008).

6.2 Empirical Results

6.2.1 Grid Overheads

We have performed some experiments to get an idea of the overheads discussed in Sect. 6.1. We have used a dummy optimization problem so that we can change all the parameters according to our needs. The dummy optimization problem in use is a simple one-max optimization problem. However, unlike a simple one-max problem we can control the time (T_f) taken by a single objective function call. We try to map the results on Eq. 6 to find the overheads. As resources, GridUFO has 16 dedicated IBM x3455 AMD® Dual-Core Opteron model servers (with 2GB of memory) as Globus based resources, and one Condor pool with 16 execute resources of similar specifications. These resources can be arranged into logical topologies by modifying a topology configuration file. New resources can be added very easily by modifying this configuration file.

Table 3 shows the empirical results obtained by running the dummy problem over GridUFO using a "Simple Real GA" as the optimization algorithm. T_s is the time taken by serial implementation of the code, T_p is the time taken by the parallel implementation over GridUFO, T_f is the time taken by a single fitness value calculation. T_s/T_p is the Speedup achieved by implementation over GridUFO, m is total number of clusters involved, u^{avg} is the average time taken by a cluster to execute $(1/m)^{th}$ part of the algorithm, and O_{inter} is the average communication overhead per cluster. We have used 16 clusters for the implementation and each cluster consists of only one IBM x3455 AMD® Dual-Core Opteron model servers (with 2GB of memory) Globus based resource. Each cluster has only one computational node, as we only want to observe the inter cluster overheads (O_{inter}). Value of m is 16 throughout the experiments. Moreover, population size is 90 and maximum generations are set to 10. Call to the fitness function was made 990 times by the algorithm. Each result is an average of 10 independent runs with similar conditions and pa-

rameters. Time for initialization and destruction of GridUFO API is also included; however, time for scheduling the job is not considered as it can vary in wide ranges.

It is clear from the table that the overheads due to the implementation over a Grid are quite significant for smaller problems. However, we can obtain considerable speedups for the problems of large sizes. We increase the execution time of single objective function call to simulate a larger or a harder problem. Therefore, in terms of speedup GridUFO and other such environments are more suitable for the problems of relatively large sizes.

7 Conclusions & Future Work

In this paper, we demonstrated a unified approach for constructing an optimization PSE using an OGSA based Grid middleware. This SOA compliant framework works in consent with the true spirit of Grids and offers all its functionality as WSRF services to the users. GridUFO eradicates the short comings of the earlier projects in the area and also provide some extra features. Emphasis has been given to the ease of use without compromising on the flexibility and functionality provided by the Grid. Using GridUFO a user can either solve an optimization problem using the optimization algorithms available in the framework, or add a new algorithm to the framework and share with other GridUFO users. Similarly the user can either use an existing objective function (optimization problem), or add a new one and share with other users. This strategy allows the user to use any algorithm to solve the optimization problem at hand, or the user is allowed to use same algorithm to solve different optimization problems. User is not required to install any software as the framework is accessible through Web portal (can be used through any modern Web browser). Moreover, the Web service provided with the framework can be consumed directly from within the client application independent of the programming language and environment used by the client application. The work presented in this paper can act as a foundation work for other similar projects.

In our experience the learning curve of Grid is very gradual, and therefore, many application developers are either reluctant to use Grid or waste a lot of time in learning the basics of Grid before starting with actual implementations. There is a dire need for more Grid based PSEs similar to GridUFO, to attract more and more application developers to this computing paradigm of the future.

In future areas of work can be improvement of QoS mechanism in order to improve the framework. Adding an economically feasible utility computing model to the system can be a good contribution. Introduction of advanced resource reservation is also a hot topic for research. Improvements can be made to the Web portal in order to make it more user friendly. We plan to make the next version of GridUFO API that will allow the user to use two or more algorithms in a hybrid fashion over

distributed computing environments like Grid.

8 Acknowledgments

We would like to acknowledge a number of people who have helped us with the problems at different parts of this project. We would like to thank Hidemoto Nakada, Masato Asou, Yoshio Tanaka for their help, regarding the use of Ninf-G. We would also like to thank Hiroshi Takemiya for sharing his knowledge about the Grid middlewares and other tools. We thank Michael Russell and Jason Novotny for their help on resolving GridSphere related problems. We also thank the anonymous users who replied to our queries on the mailing lists.

References

[1] Optimization service provider, http://www.osp.org/.

[2] The vine toolkit project, http://gforge.man.poznan.pl/gf/project/vine/.

[3] D. Abramson, A. Lewis, T. Peachy, Nimrod/o: A tool for automatic design optimization, in: The 4th International Conference on Algorithms & Architectures for Parallel Processing (ICA3PP 2000), Hong Kong, 2000.

[4] E. Alba, B. Dorronsoro, Solving the vehicle routing problem by using cellular genetic algorithms, in: EvoCOP, 2004.

[5] E. Alba, J. Garcia-Nieto, A. Nebro, On the configuration of optimization algorithms by using xml files, Tech. rep., The TRACER Project, http://tracer.lcc.uma.es/ (March 2003).

[6] J. Almond, D. Snelling, Unicore: Uniform access to supercomputing as an element of electronic commerce, in: Future Generation Computer Systems, 1999.

[7] G. Amdahl, M. Gene, Validity of the single processor approach to achieving large scale computing capabilities (2000) 79–81.

[8] S. Cox, L. Chen, S. Campobasso, M. Duta, M. Eres, M. Giles, C. Goble, Z. Jiao, A. Keane, G. Pound, A. Roberts, N. Shadbolt, F. Tao, J. Wason, F. Xu, Grid enabled optimisation and design search (geodise), in: UK e-Science All Hands Meeting, Sheffield, UK, 2002.

[9] J. Czyzyk, M. Mesnier, J. More, The neos server, IEEE Journal on Computational Science and Engineering 5 (1998) 68–75.

[10] E. Dolan, The neos server 4.0 administrative guide, Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory (May 2001).

[11] I. Foster, Service-oriented science: scaling the application and impact of eresearch, in: First International Conference on e-Science and Grid Computing, 2005.

[12] I. Foster, Globus toolkit version 4: Software for service-oriented systems, in: IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, 2006.

[13] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufman, 1999.

[14] G. Fox, M. S. Aktas, G. Aydin, H. Gadgil, S. Pallickara, E. Pierce, A. Sayar, Algorithms and the grid, in: Computing and Visualization in Science (CVS), 2005.

[15] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke, Condor-G: A computation management agent for multi-institutional grids, in: Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC), San Francisco, California, 2001.

[16] W. Gropp, J. Mor'e, Optimization environments and the neos server, in: M. D. Buhmann, A. Iserles (eds.), Approximation Theory and Optimization, Cambridge University Press, 1997.

[17] J. Herrera, E. Huedo, R. Montero, I. Llorente, A grid-oriented genetic algorithm, in: Advances in Grid Computing - EGC 2005, 2005.

[18] H. Imade, R. Morishita, I. Ono, N. Ono, M. Okamoto, A grid-oriented genetic algorithm for estimating genetic networks by s-systems, SICE 2003 Annual Conference 3 (4-6) (2003) 2750–2755.

[19] H. Imade, R. Morishita, I. Ono, N. Ono, M. Okamoto, A grid-oriented genetic algorithm framework for bioinformatics, New Generation Computing 22 (2) (2004) 177–186.

[20] Y. Ishikawa, Y. Kaneo, M. Edamoto, F. Okazaki, H. Koie, R. Takano, T. Kudoh, Y. Kodama, Overview of the gridmpi version 1.0, in: Summer United Workshops on Parallel, Distributed and Cooperative Processing, SWoPP05, 2005.

[21] S. Larson, C. Snow, V. Pande, Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology, R. Grant, ed, Horizon Press, 2003.

[22] G. Laszewski, The Java CoG Kit User Manual, version 4.0, Mcs technical memorandum, Argonne National Laboratory, Mathematics and Computer Science Division, 9700 S. Cass Ave, Argonne, IL 60439, U.S.A. (March 2004).

[23] D. Lim, Y. Ong, Y. Jin, B. Sendhoff, B. Lee, Efficient hierarchical parallel genetic algorithms using grid computing, Future Generation Computer Systems 23 (4) (2007) 658–670.

[24] F. Lobo, G. Harik, Extended compact genetic algorithm in c, Tech. Rep. IlliGAL Report No. 99016, Urbana, IL: University of Illinois at Urbana-Champaign. (1999).

[25] K. Miura, Overview of japanese science grid project: Naregi, in: Progress in Informatics, 2006.

[26] A. Munawar, M. Wahib, M. Munetomo, K. Akama, Standardization of interfaces for meta-heuristics based problem solving framework over grid environment, in: Proccedings of HPCAsia 2007, Seoul, South Korea, 2007.

[27] A. Munawar, M. Wahib, M. Munetomo, K. Akama, Linkage in Evolutionary Computation (to appear), chap. Parallel GEAs with Linkage Analysis over Grid, Springer, 2008.

[28] M. Munetomo, Realizing virtual innovative laboratory with robust evolutionary algorithms over the grid computing system, in: Proceedings of the 6th International Conference on Recent Advance in Soft Computing, 2006.

[29] M. Munetomo, D. Goldberg, Identifying linkage by nonlinearity check, Tech. Rep. IlliGAL Report No. 98012, Urbana, IL: University of Illinois at Urbana-Champaign. (1998).

[30] M. Munetomo, N. Murao, K. Akama, A parallel genetic algorithm based on linkage identification, in: Genetic and Evolutionary Computation GECCO 2003, Springer, 2003.

[31] J. Novotny, M. Russell, O. Wehrens, Gridsphere: a portal framework for building collaborations, Concurrency and Computation: Practice and Experience 16 (5) (2004) 503–513.

[32] M. Pelikan, D. E. Goldberg, E. Cantú-Paz, BOA: The Bayesian optimization algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, vol. I, Morgan Kaufmann Publishers, San Fransisco, CA, Orlando, FL, 1999.

[33] K. Symour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, H. Casanova, Overview of gridrpc: A remote procedure call api for grid computing, Proc. 3rd Int. Workshop Grid Computing (2002) 274–278.

[34] H. Takemiya, Y. Tanaka, S. Sekiguchi, S. Ogata, R. Kalia, A. Nakano, P. Vashishta, Sustainable adaptive grid supercomputing: multiscale simulation of semiconductor processing across the pacific, in: SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, ACM, New York, NY, USA, 2006.

[35] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, S. Matsuoka, Ninf-g: A reference implementation of rpc-based programming middleware for grid computing, Journal of Grid Computing 1 (1) (2003) 41–51.

[36] D. H. Wolpert, W. G. Macready, No free lunch theorems for search, Tech. Rep. SFI-TR-95-02-010, Santa Fe, NM (1995).