Distributed Web Based Computation (DWBC)

Z. Goldberg, M. Vainstein, and M. Bercovier

School of Computer Science and Engineering
The Hebrew University of Jerusalem 91904, Israel
E-mails: zvikig@cs.huji.ac.il, maxbetta@cs.huji.ac.il, berco@cs.huji.ac.il

Key words: MATLAB, SCILAB, GRID, Parallel, Distributed, Scheduling, SOA

Abstract
The present work introduces the design of Distributed Web Based Computation (DWBC), which is the next generation of the GBPM (Grid Based Parallel MATLAB) project. The GBPM provides infrastructure for handling MATLAB computation inside a cluster or a grid, whereas the new system takes this approach one step further. It deals with massive computation of any kind, internet based distribution and pluggable algorithm for finding available remote servers.

DWBC is written as SOA (Service Oriented Architecture) and therefore robust, easy to use and portable. The main goal is to enable distributed way for handling massive computation needs from the Internet (not only inside a cluster/grid) as well as many aspects of parallel computing such as asynchronous, transparency and fault tolerance. Additional aspects include a friendly user interface, an efficient distribution algorithm, and secure environment.

The present development has been applied to SCILAB [24] using gossip algorithm for passing essential data for finding the available remote server, however, it can easily support different software or algorithm.

The current implementation consists from 3 elements: DWBC SCILAB Client (Client interface for sending SCILAB scripts), DWBC Server (handles the server SCILAB computation) and UDCD (Universal Distributed Computation Discovery) for discovering available DWBC servers.

The implementation is based on Microsoft technology (.NET Framework 2.0), Microsoft Smart Client technology at the client end, Web Services at the Server end.
1 Introduction

It becomes more and more common to use scientific software for various scenarios. Many of those scientific software's available today provide flexibility, graphics, virtualization and high level programming. All were originally designed for local computation.

The need for large computation is growing rapidly. One solution is to use so-called ‘super-computers’ and very large clusters. The difficulty might be handling shared or distributed memory. The issue is that many CPUs need fast access to memory and therefore will likely use a cache memory, which might cause data to be incoherent.

Another trend is to use the ever extending networking capabilities. Hence distributed remote execution is an emerging option.

In that category Grid computing is definitely one of the interesting options. It provides an emerging computing model with the ability to perform higher throughput computing by taking advantage of many networked computers to model a virtual computer architecture that is able to distribute process execution across a parallel infrastructure.

In this paper we present the goals, design, implementation and performance evaluation of the Distributed Web Based Computation (DWBC), which is the next generation of the GBPM - Grid Based Parallel MATLAB toolbox [1,2]. The latter provides infrastructure for handling MATLAB computation inside a cluster or a grid.

The new system takes this approach one step further. It deals with massive computation of any kind, internet based and pluggable algorithm for finding available remote servers using the latest technology such as Service Oriented Architecture (SOA) and Microsoft .NET framework 2.0.

The implementation is on SCILAB, which provides high level programming language, loosely based and similar in functionality to MATLAB and is freely available to download. However, as it written in this paper, the design should support MATLAB or any other scientific software.

This paper is organized as follows: in the 2nd section we survey the related works existing for MATLAB and SCILAB, the 3rd introduces DWBC approach, the 4th describes the design of DWBC project, the 5th section details the implementation issues, the 6th section presents the performance evaluation of the main routines and the 7th section describes possible future work.
2 Related Work

This paper describes the infrastructure for activating tasks remotely (including discovery, scheduling, load balancing etc') from any scientific software. A few projects dealt with those issues along the years. For instance the Condor project [22, 23] is a high-throughput distributed batch computing system. Condor provides a job management mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Unlike those projects, our project is not a batch system. It sends and receives tasks online and limited to a specific application (it can be easily limited to more applications). Another important goal is to provide an infrastructure to be used anywhere anytime. That is why we choose the grid computing model [10] (over the internet).

We briefly describe two of the well known scientific software and the approaches those companies chose.

SCILAB approach [3, 7, 19, 20 and 24]:
SCILAB, remote execution project, is called ProActive [7]. Its goal is to equip SCILAB with a generic interface to Grid computing. Main features are:

- Remotely accessible Objects (RMI, JINI)
- Asynchronous Communications
- Group Communications, Migration (mobile computations)
- XML Deployment Descriptors
- Interfaced with various protocols: rsh,ssh,LSF,Globus,---> SOAP
- Visualization and monitoring: IC2D

This extension allows the deployment of SCILAB instances on several nodes of the grid (and to use these instances like computing engines) and the submittal of SCILAB tasks over the grid. Those nodes are called virtual nodes and are mapped and configured in an XML file at the client end.

ProActive project uses portable libraries such as PVM [16] (Enable users to send data of any type to remote Scilab instance in order to solve much larger problems at minimal additional cost), BLAS, LAPACK and ScaLAPACK [18].

MATLAB approach: (For further reading [4, 11, and 14]).
MATLAB implemented the remote execution differently from SCILAB. The agent, which executes the scripts remotely, is called worker, whereas the discovery of a worker is done via a scheduler or job manager. When a new worker wants to be activated, it needs to be registered in a specific scheduler. A worker can be signed to only one scheduler. The scheduler discovery is done manually by its name.
MATLAB client, which is installed on a desktop PC, is sending jobs to a scheduler or job manager, which passes the task to a worker. The workers evaluate tasks one at a time, returning the results to the job manager. The job manager then returns the results of all the tasks in the job to the client session.

The infrastructure uses an MPI library [17] and based the parallel execution around it. By that, it enables users to parallelize functions. The key concepts are:

- **Message passing** – using MPI [17] standards to send messages to the remote machines. By using this library, MATLAB users are able to take advantage of its power [5] (efficient communication, reliable communications interface, handling communication failures etc’).

- **Backend support** - MATLAB is in the front end for a parallel computation engine. Computation made on the engine uses some high performance numerical computation libraries like ScaLAPACK [6].

- **Compilation** - In this approach MATLAB scripts are compiled into a native parallel code, sometimes translated to C or FORTRAN as an intermediate stage. The advantage that the compiled code runs without the overhead incurred by MATLAB and allows the produced parallel program to run on platforms which does not support MATLAB.

- **Embarrassingly parallel** - This approach exploits SPMD3 paradigm reference - a parallel programming model in which different processors execute the same program on different sets of data.
3 DWBC Approach

Remote execution is not something new. Many different implementations already exist today [3, 4, 7, 9, 10, 11]. The aim of the DWBC project, is to make an existing stand alone software to easily remote execute scripts within a given application software.

The implementation is based on proven industry standards such as Web services, SOAP [15] messages, Service Oriented Architecture (SOA), Object Oriented and Polymorphism and Asynchronous communication. Therefore, in developing our project we had in mind:

- Ease of use / friendliness – Keep the client software as it is. The user must remain is his own environment and the programming style should be similar to the software style. In addition, the remote execution must be transparent to the end user. In this way, the end user will parallel its problems easily.
- Grid computing – provides the ability to perform higher throughput computing by taking advantage of many networked computers available throughout the internet.
- Load balancing – critical issue of parallel computation on heterogeneous network that must be addressed. Because we are talking about Grid computing over the web, we cannot take this term obviously. Inefficient load balancer will reduce the impact of remote execution in grid environment.
- Generic infrastructure – provides a comprehensive API to simplify the programming in DWBC environment. Adding software should be easily and straight forward.

---

1 SOAP (originally Simple Object Access Protocol) is a protocol for exchanging XML-based messages over computer network, normally using HTTP. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework which other abstract layers can build on.
4 DWBC Design

4.1 General overview

Using industry proven architecture such as client-server and service oriented architecture (SOA), enables us to provide a distributed computation service and a client that will use this service. The message protocol between the client and the server are based on SOAP [15] messages over HTTP. The infrastructure here developed provides the necessary interfaces for distributed computing the DWBC client and server. In addition to the infrastructure, to illustrate our work we give an example with Scilab, that is we implement a client integration of Scilab to our infrastructure and a server integration from the remote server to a remote Scilab.

This architecture provides us with the ability to develop another client which will be able to use the existing server (and vice versa).

*Project high-level architecture*

The high level architecture is shown below.
**DWBC Client** – Enables the client software to execute scripts remotely. This project include a Scilab implementation, however, it can be done quite easily for any other scientific software. The client is responsible for:

1. **Parallelization** – Enables the client software (Scilab for example) to remote execute script asynchronously. Manages the process all along from getting the request until sending the response from the remote server.
2. **Discovery of DWBC servers** – This is one of the most important issues. This will be discussed in details and is done using UDCD (Universal Distributed Computation Discovery) and gossip algorithm implementation.
3. **Monitoring** – Provide the information about each and every remote script. The details include the time, server details etc'. Errors, warnings and general information are gathered and written in a central log.

**DWBC Server** - Receives DWBC client scripts and execute it remotely. As stated here above, it is done for Scilab; however, it can be implemented quite easily in any other scientific software. The server is responsible for:

1. **Execution** – Executes the scripts received from clients. The server raises the specific service and passes the necessary arguments. The server is also deals with nearby aspects like Error handling, asynchronous activation etc'.
2. **Discovery of DWBC servers** – New server can be registered in the UDCD (Universal Distributed Computation Discovery) or can get some existing servers from it. Any server on the network is participating in the discovery job. This will be discussed later in details.
3. **An Important aspect which exists both in the client and in the server is their genericity.** The architecture was designed to support any scientific software with minimum adaptation. The design, which will be discussed in the next chapter, is taking advantage of inheritance, interfaces, polymorphism (characteristic of Object Oriented Programming) etc’. As written before, the project is implemented in Scilab, however, it can be easily adapted to different scientific software. The implementation uses 'DWBCServices' API (which will be discussed later), which provides all the necessary tools for remote execution in the Distributed Web Based Computation environment.

### 4.2 Resource Allocation

In order to execute remote scripts, the client has to allocate somehow a remote server. When dealing with a cluster inside a network, this problem might not exist. However, when talking about grid computing, it is another issue. There are basically
two main approaches. Centralized resource control and the localized resource control. The first one, is not scalable, has one point of failure and not fault resilience. The last, leads to unstable resource allocation.

We mix both approaches. We implement UDCD (Universal Distributed Computation Discovery) service, which is XML based registry. A computer can register to the service or can get a list of the registered servers. Another way for receiving information is from previous execution or from the gossip algorithm (which will be discussed next).

![Diagram of UDCD service](image)

4.3 Load balancing

Load balancing is a technique to spread work between many computers, processes, disks or other resources in order to get optimal resource utilization and decrease computing time. In order to do so, the load balancer must be aware of the load of every single machine. With that, the load balancer will be able to select the appropriate server to run the job. It can be the server which has the lowest load, the nearest or any other. Inside a network, when the entire computers are located in one cluster, this issue is not as complex as when considering grid computing over the internet. When talking about that, we must deal with extra complex issues such as: heterogeneous networks, networks that changes continuously as new nodes join and old nodes leave the network, distributed dynamic discovery, reliability and scalability.

We do not want central control for maintaining the balance. Rather, we seek mechanisms by which nodes communicate in a relatively homogeneous fashion with one another, so as to spread information updates.
Hence, we use random gossip [8, 21] as the algorithm for transmitting the information. The main idea in the gossip algorithm is that in each step, each node (server) chooses a different node (server) at random, and forwards all the information it knows about. A well-known result states that with high probability, all servers will receive a copy of the information within $O(\log N)$ steps of its initial appearance [12, 13].

DWBC implementation provides an API for controlling gossip algorithm. The clients and the servers gossip all the time with each other passing the necessary information. When the user opens the DWBC client, the gossip is started (as shown in the figure below). The client seeks servers to exchange information with. When a transmittal is requested to a remote server, the client chooses the appropriate server from its internal information.
5 Implementation issues

5.1 DWBC software implementation

The client and the server software architecture are quite similar. Both of them rely on the same foundations, however, with some differences. Each layer, which will be described below, may communicate with its adjacent (up and down layer)

We will review the software architecture from the bottom to the top.

Windows Operating System/.Net framework 2.0 – This is the lowest layer. DWBC was implemented on Microsoft windows platform and requires .NET Framework 2.0. However, the DWBC software design can be applied on any operating system and of course any programming language.

DWBC Services API – This layer warps the pervious layer and provide an extensive API to support all operation needed by the client or server. The services implements the gossip algorithm (used both by the client and the server), the message passing protocols (which are TCP protocol between Scilab application and the DWBC client and HTTP and SOAP protocols between the DWBC client and DWBC server). This API is used by the DWBC client and the DWBC server and can be used by any other implementation.

DWBC Client – This layer is an implementation based on the DWBC services API. This layer is stateful and provides extra information such as logging, GUI (Graphical User Interface) etc'.

Scilab Software – The top layer communicates with the DWBC Client layer. The layer transmits messages to the DWBC client layer and waits for a response.

<table>
<thead>
<tr>
<th>Scilab</th>
<th>DWBC Client</th>
<th>SOA (Web Services)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWBC Services API</td>
<td>Scilab Agent</td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>SOAP Messages</td>
<td>Gossip Algorithm</td>
</tr>
<tr>
<td></td>
<td>Net Framework 2.0</td>
<td>Windows OS</td>
</tr>
</tbody>
</table>
5.2 SCILAB integration

The integration with SCILAB as well as integration with any of computation engines implements 3 tiers:

1. Public API Integration that provides a set of functions that should be called by engine users in order to reach DWBC.
2. Business logic is a core that implements all relevant data management and event handling mechanisms.
3. Communication tier is a network based component that bridges between business logic to DWBC local service (DWBC client).

All three tiers are implemented in a single DLL based on C++ technology where the Public API tier implements an interface that matches communication's engine external module’s standards.

5.2.1 Public API

Public API implements a number of functions:

1. SubmitJob() is an asynchronous function that provides a function that is called to submit a Job.
2. WaitForAll() is a synchronous function that provides a function that is waitable and releases when all (or a subset of) submitted jobs are finished.
3. Reset() is a synchronous function that provides a function that resets all previously submitted jobs.
4. GetStatus() is a synchronous function that provides a function that returns a status of specific submitted job.

5.2.2 Business Logic

Business logic implements a number of data bases for job submissions management and a set of callback functions that used to be called on events for example job reply received.

In addition, Business Logic implements a set of function that analyze Job requests and prepare them as a package that will be executed on remote machine. Such a package includes:

1. Command line
2. Depends on files that implements function that are not build-in functions.
3. Environment file that stores relevant variables that might be needed or changed during remote execution of the job

5.2.3 Communication Tier

The communication tier implements LPC protocol and provides a communication channel between DLL-based integrated module to DWBC local service.

6 Future work

The future work might be in some different direction. The most intuitive way is to add new scientific software. As explained in this paper, the DWBC infrastructure can be easily extended to support other scientific software. Another idea might be extending the way that chooses the server to send the execution to. Now, the algorithm is pretty simple and the decision is based on the average CPU usage of the remote computer. A good extension might be improving the load balancing. The algorithm can take into consideration some extra details such as amount of CPUs, available memory size, CPU usage in the recent time, idle time on the remote server etc'. Totally different ideas might be improving the security on the remote server and enable processes migration.
7 REFERENCES

[18] Ed F. D’azevedo and Jack J. Dongarra, The design and implementation of the parallel out-of-core ScalaPack LU, QR and Cholesky Factorization routines, Oak Ridge National Laboratory and University of Tennessee, Knoxville
[19] Laurent Baduel, Françoise Baude, Denis Caramel, Efficient - Flexible and Typed Group Communications in Java, Nice Sophia Antipolis 2004
[20] Romain Quilici - ProActive Architecture of an Open Middleware for the Grid,  
http://www.objectweb.org/ProActive

International Computer Science Institute, Berkeley and University of Massachusetts, Amhest.

[22] Douglas Thain, Todd Tannenbaum, and Miron Livny - Distributed Computing in Practice: The Condor Experience, University of Wisconsin-Madison, 2004

