

# A Multi-Level WEB Based Parallel Processing System

## A Hierarchical Volunteer Computing Approach

Abdelrahman Ahmed Mohamed Osman

### I. INTRODUCTION

**Abstract**— Over the past few years, a number of efforts have been exerted to build parallel processing systems that utilize the idle power of LAN's and PC's available in many homes and corporations. The main advantage of these approaches is that they provide cheap parallel processing environments for those who cannot afford the expenses of supercomputers and parallel processing hardware. However, most of the solutions provided are not very flexible in the use of available resources and very difficult to install and setup.

In this paper, a multi-level web-based parallel processing system (MWPS) is designed (appendix). MWPS is based on the idea of volunteer computing, very flexible, easy to setup and easy to use. MWPS allows three types of subscribers: simple volunteers (single computers), super volunteers (full networks) and end users. All of these entities are coordinated transparently through a secure web site. Volunteer nodes provide the required processing power needed by the system end users. There is no limit on the number of volunteer nodes, and accordingly the system can grow indefinitely. Both volunteer and system users must register and subscribe. Once, they subscribe, each entity is provided with the appropriate MWPS components. These components are very easy to install.

Super volunteer nodes are provided with special components that make it possible to delegate some of the load to their inner nodes. These inner nodes may also delegate some of the load to some other lower level inner nodes .... and so on. It is the responsibility of the parent super nodes to coordinate the delegation process and deliver the results back to the user.

MWPS uses a simple behavior-based scheduler that takes into consideration the current load and previous behavior of processing nodes. Nodes that fulfill their contracts within the expected time get a high degree of trust. Nodes that fail to satisfy their contract get a lower degree of trust.

MWPS is based on the .NET framework and provides the minimal level of security expected in distributed processing environments. Users and processing nodes are fully authenticated. Communications and messages between nodes are very secure. The system has been implemented using C#.

MWPS may be used by any group of people or companies to establish a parallel processing or grid environment.

**Keywords**—Volunteer computing, Parallel Processing, XML-Web Services, .NET Remoting, Tuplespace.

**D**ESPITE the dramatic increase in computer processing power over the past few years [8], the appetite for more processing power is still rising. The main reason is that: as more power becomes available, new types of work and applications that require more power are generated. The general trend is that new technology enables new applications and opens new horizons that demand further power and the introduction of some newer technologies.

According to [2], developments at the high end of computing have been motivated by numerical simulations of complex systems such as:

- Simulation and Modeling problems
- speech recognitions training
- Problems dependent on computations / manipulations of large amounts of data
- Image and Signal Processing
- Entertainment (Image Rendering)
- Database and Data Mining
- Seismic
- Climate Modeling
- Human Genome

However, there are indications that commercial applications will also be in demand for high processing powers. This is mainly because of the increase in the volumes of data treated by these applications.

There are two main approaches to increase computer processing power:

**Improving the processing power of computer processors:** This can be achieved by decreasing the clock cycle of the processor and optimizing the way instructions are executed. The clock cycle of the processor is the time required to execute the most primitive operation. There is evidence now that processor clock cycles are decreasing slowly and approaching their physical limit (the speed of light) [3]. Instruction execution optimization is achieved using some techniques such as pipelining.

**Using multiple processors to perform computations:** Multiple processors can use a shared memory (multiprocessor systems) or independent memory, where each processor is equipped with its own RAM (multi-computer systems). This

approach is very promising and has known physical limit. Multiprocessor and multi-computer systems are the building blocks of parallel processing systems.

The traditional approach to build parallel processing systems is to build a single-box computer with one or more boards, each equipped with a number of processors. However, another rising approach is to use computer network technologies to build loosely coupled parallel systems.

Computer network capabilities improved dramatically over the past few years, both in speed and reliability. While the speed of early computer networks is only 1.5 Mbits per second, the speed of most current computer network exceeds 1000 Mbits per second. The use of optical fiber technology greatly improved the reliability of computer networks and made them as reliable as standalone systems. This implies that a wide variety of flexible parallel systems can be built using computer network technologies.

Observing this, a number of efforts has been made to build loosely coupled parallel systems using computer network technologies. One of the first well known efforts is the parallel virtual machine [1]. A number of similar projects have been initiated in different places. These systems represent the first generation of loosely coupled parallel systems.

The second generation of loosely coupled systems relied on distributed technologies such as DCOM, and CORBA [4]. These technologies are object-oriented and provide interfaces for communication between distributed objects.

## II. .NET

In this paper, the newly emerging .NET technology [5] is used to build a multi-level loosely parallel processing system. .NET provides a mature, reliable, secure development environment that can be used to build distributed applications of any kind. Although, many options are available in the .NET framework, web services and the .NET remoting are the two techniques chosen to build the system. This is mainly because of their power and wide acceptability. Web services are based on the standard SOAP protocol and the .NET remoting provides a reliable replacement for the classical RPC technology.

In our system, web services are used at the top level, while .NET remoting is used at lower inner nodes.

The security and reliability features of the .NET framework have been used to strengthen the system and close all security gaps.

In addition, the system uses a simple behavior-based scheduling algorithm to distribute and execute processing tasks.

## III. MWPS COMPONENTS

The MWPS system consists of four software components, three components are downloadable from the web site after user registration. The fourth one (the main part) is installed in the server which represents the web site that contains other components and Task-coordinator to work with these

components (Fig. 1).

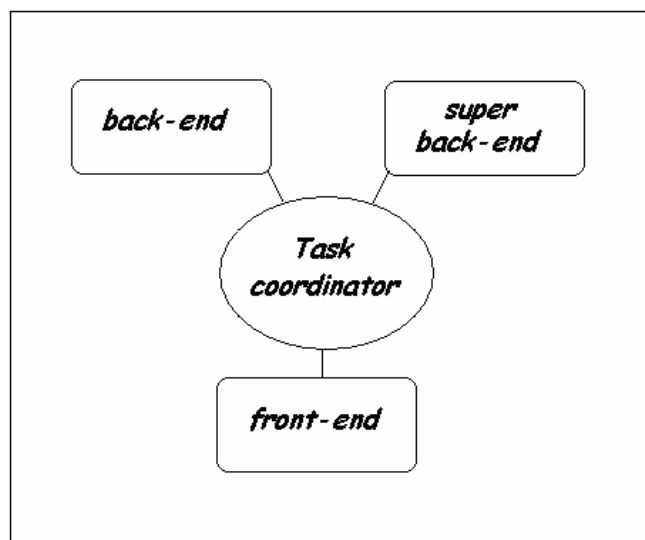


Fig. 1 MWPS main components

The three downloadable components are:

The front-end component: This is a simple visual tool used to submit parallel processing applications to the system. The tools allow the user to upload the code, configure the participant nodes and perform other tasks.

The back-end component: This component is installed on ordinary volunteer nodes. It enables volunteer nodes to get tasks from coordinators and return results to user nodes.

The super back-end component: This component is installed on super volunteer nodes. It enables these nodes to get tasks from coordinators, distribute tasks among their lower level nodes.

### Note:

The difference between the two types of back-end components is that the first type represents single computer resources that volunteered to perform computations in MWPS, while the second represent a special type of volunteer node which may be full LANs or Multi-level LANs. This type of node may take a full parallel processing application and distribute it among its local nodes.

### A. Front-end Component

This component represents a visual tool used to submit parallel processing applications; these applications are executed by volunteer's remote machines (back-ends). The system interface makes accessing remote resources as easy as accessing local ones. Thus, when the user uploads new tasks, the Task Coordinator puts them in MWPS tuplespace [7], available volunteer machines (back-ends) can get tasks from tuplespace using a special scheduling strategy.

The visual tools enable the user to upload new tasks, view the progress of his current tasks. When the application first starts, the user is prompted to log in (Fig. 2).

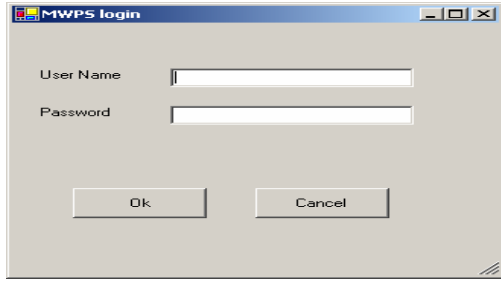


Fig. 2 MWPS login form

If the user can be successfully authenticated the user's main menu appears (Fig. 3).

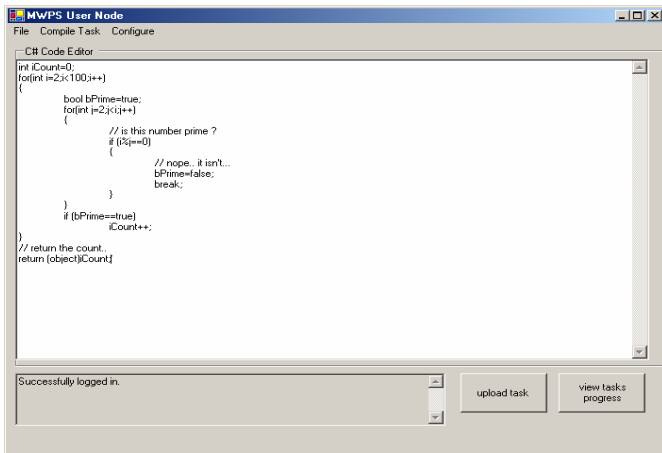


Fig. 3 MWPS user client in action

When the user writes her/his code, s/he can upload it to the tuplespace in the server by clicking upload task button (which will compile the code before uploading it). The user can logout from the system by choosing exit from file or simply by closing main menu.

### B. Back-end Component

On the server side, a Task Coordinator performs two main jobs:

- Receive Tasks from System user
- Put tasks in tuplespace

The database represents a tuplespace, where the new tasks (received tasks) are inserted in the tuplespace using a web method uploadTask() (like out() used in Linda[6] to store tasks in tuplespace), volunteers send requests for the tuplespace by calling a web method called downloadTasks(). This method checks for available new tasks (this looks like in() function used in Linda for retrieving tasks from tuplespace).

The protocol of communication between the volunteer (back-end) and Task-Coordinator to execute tasks is shown in (Fig. 4).

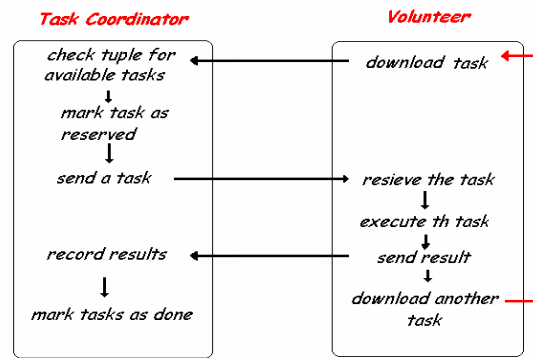


Fig. 4 Volunteer-Task-coordinator protocol

### C. Storing Results

When a volunteer finishes executing a task, it sends the result to the task coordinator and the task coordinator records the results in a tuplespace with other information that is used later by the scheduler for improving system performance. The information includes:

- The task results
- User ID (who executed the task)
- Time spent in executing the task
- Number of failures occurred during execution

### D. Super back-end Component

A super volunteer is special type of volunteer node, which may work on a full LAN or Multi-level LANs. This type of node may take a full parallel processing application and distribute it among its local nodes.

The main structure of the system consists of four components (Fig. 5):

- Task holder (ITask) as interface
- Task Implementer (designed to take in any object from the client that implements the ITask interface)
- Task Server
- Task Client

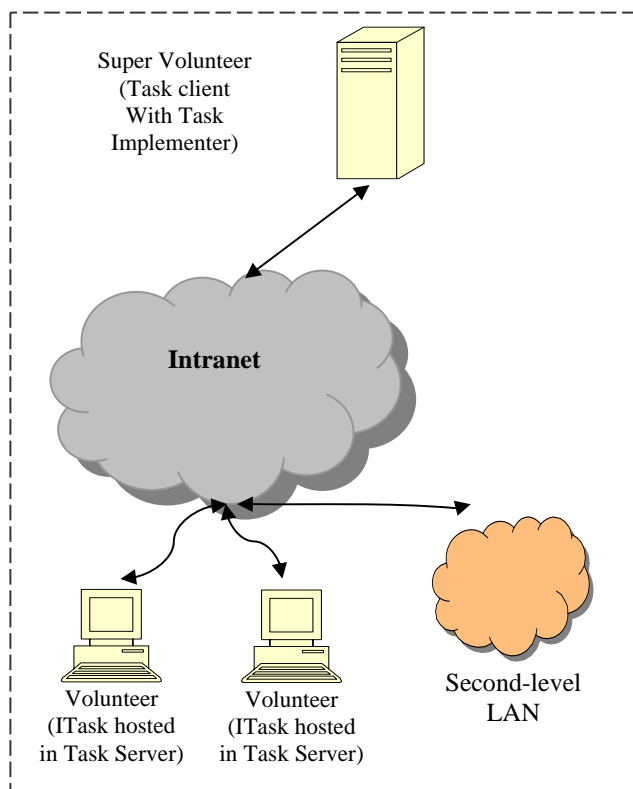


Fig. 5 Super-back end components

The connections of super volunteer with its clients in LAN is done by a piece of software that is installed while configuring the super volunteer component, this software (ITask) receives tasks from super volunteers (Task implementer), executes them, and returns the result back to super volunteer.

The protocol used between task coordinator and super back-end components is shown in (Fig. 6).

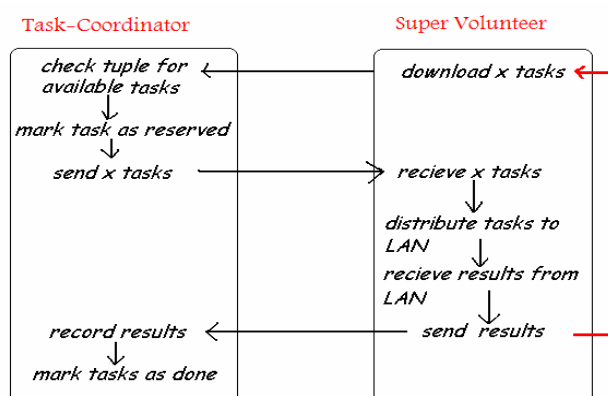


Fig. 6 Super volunteer-Task-coordinator protocol

#### IV. IMPLEMENTING MWPS

MWPS divided into two parts:

1. Server side part
2. Client side part (downloadable components)

##### 1. Server side part

The server part (Fig. 7) consists of:

Web site (registration and download components)

Task Coordinator consists of TupleSpaceComponent and a Web service

Database to implement the tuplespace

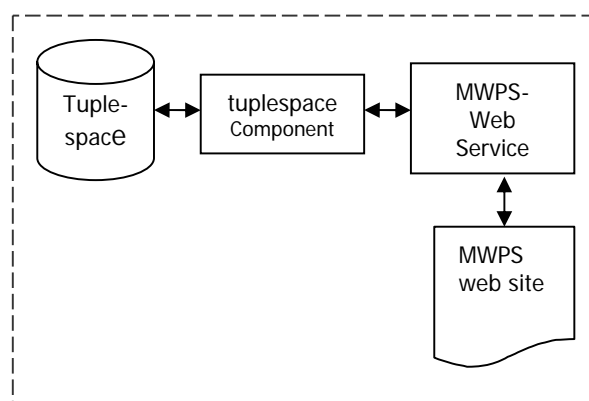


Fig. 7 Task-Coordinator

##### A. Task Coordinator

This is the main part of the MWPS and its main purpose is to distribute parallel tasks among the various volunteer nodes using a scheduling mechanism. Coordinator receives parallel processing requests from the system user and distributes these tasks to available registered volunteer nodes. The volunteer nodes return results to task coordinator when they finish. Coordinator maintains and updates database tables of all types of volunteer nodes on the system.

##### B. TupleSpace

Task Coordinator uses a database to implement the tuplespace. The tuplespace consists of four tables: Users, ActiveVol, Tasks, and Scheduler. The Users table contains the logic information for all the users who are registered in the system. For increased security, this field could hold encrypted binary information.

The ActiveVol table is used to store a record for each active volunteer. Volunteers are identified by their unique numbers. The ActiveVol table also indicates the time the volunteer is started, which is useful for scheduling and securing policies.

The Tasks table stores the tasks that have been uploaded for processing; also the status of each task and the time the task is uploaded and completed will be recorded.

Finally, the Scheduler table records all the information of

the volunteer that helps in improving system performance.

The system user provides the functionality that allows the registered users to query task information and upload new tasks.

### C. MWPS Web Service

The web service provides two functions: store tasks in tuplespace (uploading) and retrieving tasks from tuplespace (download tasks).

### 2. Client side part

This part consists of three components:

System user (front-end)

Normal Volunteer (back-end)

Super volunteer (super back-end)

These components could be downloaded from the web site after registration (Fig. 8).

The process of registration is done through the web site. The user fills his/her information, indicating his role.

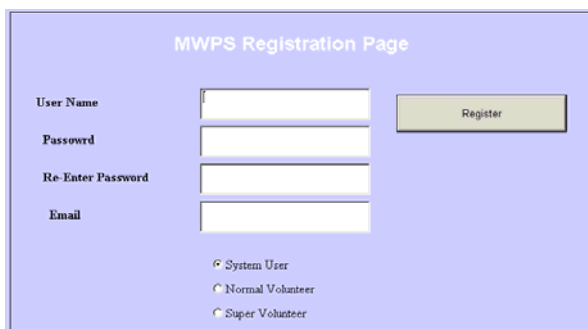


Fig. 8 MWPS registration page

When user registers successfully, s/he can download the appropriate components and start working with MWPS.

### V. THE SCHEDULING ALGORITHM

Task coordinator can receive more than one request at the same time from many volunteers to execute a task, the task coordinator check for available tasks:

If the number of tasks is more or equal to number of free volunteers, each task is sent to a volunteer, and the task coordinator waits for the results. When a given volunteer returns the results, the task coordinator checks the scheduler table for this volunteer performance if it is better than the other working volunteers that have not yet completed their tasks, it sends a copy of the task on one of the slow volunteers to the free volunteer and receives the result of this task from

the one that finishes first (replication). Also Task-coordinator sends the larger tasks to the best volunteer and the smallest one for the worst volunteers.

In case that the available tasks are less than the number of free volunteers: The task-coordinator sends tasks for the volunteers that have good history in executing the previous tasks, starting by the fastest one. Also the scheduler utilizes the scheduler information table to estimate approximately the time needed for each task and hence if any volunteer exceeds this time the task is sent again for another volunteer (the delay in returning the results may be because of crash in the system or problem in the connection...).

**Note:** Task coordinator can check the performance of each volunteer by sending a test task and calculate the execution time for each volunteer.

### VI. CONCLUSION

A loosely coupled multilevel web-based parallel processing system (MWPS) has been designed and implemented using the .NET framework. The system has been implemented using .NET remoting and web services and is based on the concept of volunteer computing. Two types of volunteer nodes are allowed: simple volunteer nodes (single machines) and super volunteers (LANS or other types of networks). The system distributes tasks between volunteer nodes. Super volunteers can further delegate the tasks assigned to them to junior nodes. In this sense the system is multilevel. Users and service providers should register before using the system. MWPS is secure and dynamic. A simple behavior-based scheduling algorithm is used to control the performance of the system.

### REFERENCES

- [1] Geist, A. et al (1994), PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing, Massachusetts Institute of Technology.
- [2] Sasikumar, M. et al (2003), Introduction to Parallel Processing, Prentice Hall of India.
- [3] aether.lbl.gov/www/classes/p139/speed/space-time.html
- [4] Lowy, J (2005), Programming .NET Components, O'REILLY.
- [5] Fedorov, A, (2002), A Programmer Guide to .NET, Addison Wesley.
- [6] <http://www.cs.york.ac.uk/linda>
- [7] <http://wiki.tcl.tk/3947>
- [8] Tanenbaum, A. AND Van Steen, M. (2002), Distributed Systems Principles and Paradigms, Prentice Hall

APPENDIX

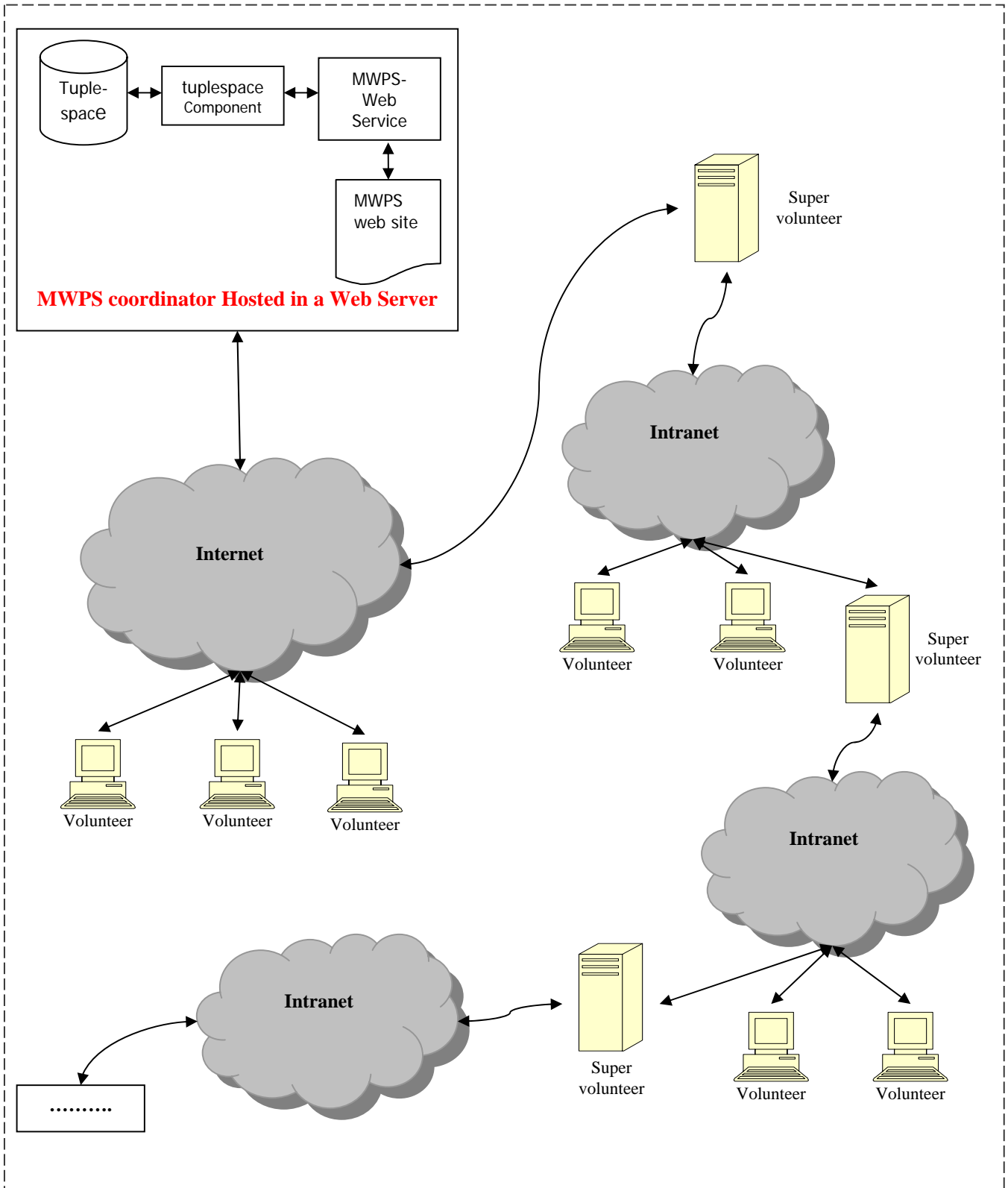


Fig. 9 MWPS main structure