Private Grid Environment for Personal Users

Heba A. Kurdi
Al Imam Muhammad Ibn Saud Islamic University, Riyadh, SA

ABSTRACT: The overall aim of this paper is to introduce the Private grid environment (PvGrid) as a ubiquitous grid environment that is not only owned and utilized by personal users but also deployed over their own devices. In this paper, architectural designs and simulation models for PvGrids are developed and evaluated. The results indicate the potential of PvGrids as ubiquitous grid environments for personal users and suggest considering deploying them of in real life scenarios.

A. Keywords - Grid computing, personal networks, mobile devices, resource scheduling.

INTRODUCTION

In grid computing [1] a set of computational resources are combined to form a large-scale distributed system in which all resources can be shared. This has the great advantage of providing a resource-rich infrastructure capable of solving data intensive and complex computational problems, such as protein folding and weather forecasting, in an acceptable time and at a reasonable cost.

However, there are two main problems with current grid systems. First of all, they are of very restricted access; they are only available for people in enterprise and research domains. In other words, personal users (individuals outside these domains) are not permitted [2, 3]. Additionally, available grid middleware systems are of very heavy weight in terms of implementations [4]. This is to say, mobile devices cannot be utilised.

On the other hand, [3] necessitated the scaling of grids to both a large number of entities and to smaller devices. There are many indicators supporting this necessity. First, every measure of the capabilities of these devices including processing speed and memory capacity, is improving, and expected to continue, at exponential rate following Moore’s law of increasing transistor density [5]. Second, the number of mobile devices in the world is escalating and expected to soon dominate the number of personal computers [6]. Indeed, the Wireless World Research Forum (WWRF) predicts that there will be 1000 wireless devices per person on average in 2017 [7,8]. Third, in many emergency situations, such as natural disasters and fire fighting, mobile devices might be the only accessible communication and computation tools. Fourth, the wireless connectivity and availability is improving as seen in current 3G networks.

Therefore, this paper aims at creating a means to bridge the gap between computational grids and personal users with resource limited mobile devices in a form of a personal mobile grid system which we called Private Grid (PvGrid).

The remainder of the paper is organised as follows. Section II proposes architectural designs for PvGrids. Section III explains the research methodology followed during this research. Section IV presents and analyses the simulation results then Section V concludes the paper.

I. PRIVATE GRID (PVGRID)

What is PvGrid?

A PvGrid is a personal mobile grid environment which can be owned and utilised by an individual user. It is constructed over his/her devices and might be extended to other devices which s/he trusts. PvGrid aim to enable the mobility of both, users requesting access to grid resources and resources that are part of a grid. Hence, the distinguishing characteristic of a PvGrid is that it is primarily constructed, owned and utilised by an individual (or a group of individuals with a mutual trust relationship). This is in contrast to traditional grids which are constructed, owned and utilised by organisations and other large entities. In other words, where traditional grids are concerned with a large user population, the PvGrid is only concerned with a single user. Also the type of application is different; where traditional grids are chiefly concerned with massive complex world-
wide computations, PvGrid applications are considerably smaller in size, scope and complexity. Additionally, where traditional grids need a well-established stationary infrastructure to operate, PvGrid can be fully accommodated in mobile devices connected via a PN.

**PvGrid Motivation**

The motivation for PvGrid is three-fold:

First, the need for grid systems which support the vision of Next Generation Grids (NGG) scaling grids to a larger number of entities and smaller devices as well as the vision of Ambient Intelligence (AmI), where humans are surrounded by computing and networking technologies unobtrusively embedded in their surroundings. Current grid architectures and technologies do not meet the requirements for turning these ambitious grid visions into reality [12, 13].

Second, the mobile device market is evolving with a progressive reduction of costs and a continuous improvement in performance, rapidly increasing the number of users and applications of such devices. The Wireless World Research Forum (WWRF) predicts that there will be 1000 wireless devices per person on average in 2017 [14]. One speculates how a personal user will be able to manage such a vast number of devices and efficiently utilise scattered resources among them. It seems reasonable to enable personal users to efficiently share resources including processor cycles, storage capacity and other functionalities among their devices in the form of services available across a global network environment such as computational grids.

Third, people are increasingly keen to frequently replace or upgrade their personal computers to gain more processing power and memory. Sometimes, they need to run complex computational jobs which their desktops or laptops cannot accommodate, or while they are on the move. People are becoming frustrated with the need to move data between their different electronic devices. Indeed, there is a need to allow users to harness all processing powers, memory storages and data files scattered across their computing and communication devices, in the form of services available across computational grids, so they can ubiquitously access data and run jobs.

**PvGrid Challenges**

There are many technical challenges in developing PvGrids. These challenges are inherited from the original components of PvGrids in three fields, as illustrated in Figure 1:

- **Grid computing**: Grid computing is a rapidly developing area of research, with heavy implementations which support neither mobile nor personal users.
- **Personal Networks**: PNs are a relatively new area of research with demanding issues such as unreliable connectivity, heterogeneity in terms of hardware and software, and high security risks.
- **Mobile computing**: Mobile computing is a challenging research area which needs to tackle problems such as resource limitation of mobile devices, low bandwidth and high dynamism.

These challenges shape the development of PvGrids more demanding than with other grids.

![Figure 1: PvGrids Challenges](image)

2.4. System Design

PvGrids are designed based on PN architecture and as a natural extension to them. A PvGrid can be viewed as a superset of PNs. It is a PN with additional resources for sharing: CPU...
cycles and run-time memories, which allow for additional public and private services.

The PvGrid layered architecture is based on the three layers (levels) PN architecture [7]. An additional layer is introduced between the network and service layers, namely the PvGrid layer. Hence, the PvGrid architecture is composed of four abstract layers: the connectivity layer, network layer, PvGrid layer and the service layer. These layers act as a middleware system offering an abstraction over physical devices.

The PvGrid layer serves as a virtualisation layer to hide the complexity of harnessing the heterogeneous underlying computational resources from the end user. In this layer, resources available from the network layer are grouped into two main categories: personal resources and foreign resources based on the type of a trust relationship between these resources.

Personal resources are grouped into larger virtual resources based on the type of functionality they provide such as CPU cycles, storage, address book and printing.

The detailed Architecture of a PvGrid consists of groups of devices which are usually owned and utilised by the same person. All these devices are connected via a well secured network PN. Issues related to connectivity are tackled in the PN connectivity layer. Issues related to security and clustering are all handled at the PN network layer, while issues related to presentation and quality of services are dealt with at the PN service layer.

Thus, basically, the key missing functional component after superimposing grid functionality on top of a PN is a resource management system for the newly added grid resources represented by CPU cycles and runtime memories, as these resources require special handling to jointly execute computational jobs in PvGrids. The main functions of this resource management system is to decompose parallel jobs, if possible, into smaller tasks that can be accommodated by mobile devices, then mapping these tasks to proper resources and, after execution, composing final results sending them back to clients. Therefore, a PvGrid includes three functional elements: clients, workers and spaces.

Typically, clients are mobile devices, such as mobile phones, usually within the PAN, that are highly dynamic and considerably limited in terms of processing power and network bandwidth. This set can send requests for executing simple jobs or complex computational jobs that are stored elsewhere, to more capable devices in PvGrid.

Workers represent devices that can be mobile but are less dynamic and have better computing resources than clients, such as laptops. These devices can jointly complete computational jobs. They are divided further into:

- **Executers**: These are computing elements capable of executing the actual computation logic encapsulated in a job after decomposing it into smaller finer grain tasks.
- **Composers**: Since jobs are decomposed into smaller tasks, and each task is executed independently of other tasks within the same job, there is a need to aggregate results produced after running these tasks. Composers are elements running a specialised program that compose all partial results related to a certain job into a final result making it ready to clients.

Spaces consist of a set of static storage-rich devices mainly at home or the office, such as desktops. Clients and workers communicate with each other using these spaces which serve basically as simple shared memories for buffering. The use of a buffering technique is important in mobile environments to reduce the impact of frequent disconnections. The idea of spaces is based on Tuple-spaces first realised in the Linda system language. A Tuple-space is a form of independent associative memory. For example, consider a group of processors that produce pieces of data and a group of processors that consume the data. Producers post their data to the space, consumers retrieve data from the space that matches certain criteria. In PvGrids, there are two types of spaces:

- **Work-spaces**: Work-spaces are multiple pools of jobs sent from clients. Executers access these pools, hunting for tasks to execute.
- **The result-space**: the result-space is a large pool holding results that are generated by executers.
Basically, two approaches are available to organise spaces. A centralised approach with a single large space, this approach has a great impact in simplifying scheduling. However, managing such a space is usually rather a challenging problem due to its massive size. Additionally, this centralised space could become a bottleneck and represent a single point of failure. The other approach is to have multiple spaces in decentralised distributed fashion. While this approach aims to solve the main disadvantages of the centralised approach, it inherits the known disadvantages of decentralised schemes represented by performance degradation and poor coordination which usually lead to a load imbalance problem.

Therefore, in this paper a new approach has been followed to avoid the shortcomings associated with previous approaches. The PvGrid design is based on multiple independent workspaces, where tasks to be executed are placed, as these spaces do not require coordination among them, as well as a single result-space where all results are buffered before being finally composed and sent to clients. The bottleneck problem in this case is easier to solve as the result-space is considerably smaller in size and lighter in traffic volume than a centralised space requiring much less management responsibility.

II. EVALUATION

The research started by surveying the area of grid computing and distributed systems for paradigms relevant to PvGrids. The survey revealed two main findings. First, there are few research projects which have addressed the mobility issue in grid computing [15] but only at the organisational level. Second, fewer research projects have targeted grid systems at the personal level [16, 17], but the focus has only been on facilitating file sharing applications. Therefore, an architectural design of PvGrids was developed to address both personalisation and mobility issues in grid computing.

The most important aspect of realising a grid system is a scheduler that efficiently utilises its resources. However, the extremely dynamic nature, diversity and limited capabilities of resources, as well as difficulties in predicting the nature and timing of incoming jobs, are all factors that increase the complexity of the scheduling problem in PvGrids.

Therefore, a survey on resource scheduling frameworks was conducted to address design features required for a resource scheduler that can cope with the extraordinarily difficult scheduling conditions in PvGrids. The survey revealed that decentralised, cooperative, local, adaptive, non-clairvoyant and self-scheduling schemes are among the top requirements to deal with the complexity of this problem. Consequently, a resource scheduler, HoPe: Honeybee inspired resource scheduler for Personal mobile grids, was proposed and implemented based on these requirements. HoPe was augmented with techniques analogous to those utilised by the honeybee colony, while allocating worker bees to nectar sources under the extraordinarily difficult conditions of weather unpredictability and food variability.

Next, PvGrid designs and HoPe implementation were evaluated thoroughly through a strictly controlled empirical study considering two main grid design issues: scalability to a larger number of nodes and sustainability under different loads.

The controlled study involved identifying the critical elements inherent in the design of grid systems and deciding on the set to be considered: job interarrival time, number of nodes, job size and processor capacity. Then varying the experimental variables, job interarrival time and number of nodes, to simulate a representative sample of grid environments. Values of the number of nodes (workers) were selected in the range of the expected number of devices per cluster available for an individual user or a small business: from 4 to 16 nodes. Values of interarrival time were selected in the range of two extreme cases of the expected usage of PvGrids: from 4 to 180 sec. after that
extraneous variables, job size and processor capacity, were controlled by randomisation to ensure a representative sample in all experiments. Heterogeneity in processor capacity was modelled assuming three types of machines \((P_a, P_b, P_c)\) with different capacities. Heterogeneity in job size was modelled assuming three types of jobs \((J_a, J_b, J_c)\) with different sizes. During running time, a uniform random number \(R_{\text{proc}}\) from one to three was generated describing the processor capacity and another random number \(R_{\text{job}}\) following the same distribution was generated to describe job size heterogeneity. Identifying a benchmark algorithm. The opportunistic scheduling heuristic (OSH), a well established heuristic in high throughput computing, was selected for this purpose. Next, suitable performance measure was identified: Speedup which refers to how much a parallel system is faster than a corresponding sequential system was used as the main performance measure 

\[
S_p = \frac{T_1}{T_p} \quad (4)
\]

where:
- \(p\) is the number of processors
- \(T_1\) is the execution time of the sequential algorithm
- \(T_p\) is the execution time of the parallel algorithm with \(p\) processors.

Although speedup is usually calculated based on one job, in the case of HoPe and OSH, calculating the speedup in this way would be out of context, as these heuristics operate in a steam of jobs. Therefore, the mean time of speedup is considered.

Finally, we compared the performance of both HoPe and OSH to optimum values and lower bounds and analysing the main findings. For improving the accuracy of this simulation-base study through: we run ten simulations and accepted the mean outcome; ignoring simulation results generated in the first 60 sec; measuring uncertainty in data using the standard deviation; and calculating absolute and relative errors to examine the quality of results.

*** III. RESULTS AND DISCUSSION ***

Simulation results obtained after running each experiment for five hours. Jobs were generated using four clients with a Poisson process and exponential interarrival times with means in the range from 4 to 180 sec. Computational jobs were implemented as divisible load cryptography applications to factor large integers (up to 4,293,001,441). Each job was contained in one packet and produced one output file. For simplicity, the communication cost was not considered at this stage. It was assumed that one machine can process only one operation at a given moment (resource constraints) and once task started, operation runs to completion (no pre-emption condition).

Figure 2 consists of three sub-figures showing the speedup of both HoPe and OSH at the three grid scales. The speedup is calculated based on equation (4) with the empirical value of 27 sec. as the execution time of the sequential algorithm. A worst bound of one is assumed, representing the case when both running times of executing a job sequentially, in one machine, and in parallel machines, are equal.

The figure shows that HoPe has maintained a noticeably higher speedup which reaches the double speedup of the OSH in nearly 60% of all scenarios. However, the difference in performance decreases gradually as the interarrival time gets larger in small and medium grid scales.

HoPe has its speedup values in the range from 2 to 10 which is double the speed of the sequential execution (worst bound) in its worst case and ten times faster than the sequential execution at best. The speedup of the OSH lies in the range from 0.9 to 7 which is a slowdown in its worst case and, in its best case, it is only seven times faster than the sequential execution.

As expected, the speedup of both HoPe and the OSH is highly affected by the grid scale in terms of the total number of worker devices in the system. The interarrival time has a lower effect when HoPe is considered.

*** IV. CONCLUSION ***

Private Grids (PvGrids) are Personal Mobile Grid environments owned and utilised by individual personal users. This paper presented architectural design of PvGrids based on PNs architectures. It carried out a controlled empirical study to experiment with the PvGrid model and
evaluate its performance in terms of speedup under different running conditions of grid scale and job interarrival times.

Experimental results indicated the dominance of PvGrid scheduler performance. These results also demonstrated the ability of PvGrid scheduler to considerably reduce the effect of variations in grid scale and job interarrival times, illustrating better scalability and sustainability, when compared to other well established schedulers. Therefore considering the deployment of PvGrids in real life scenarios is highly recommended.

REFERENCES


Figure 2: Speedup in HoPe and OSH

(a) 4 workers per cluster

(b) 8 workers per cluster

(e) 16 workers per cluster