Transparent and Reliable Computing Power Service Provision on P2P systems

Ghislain Fouodji Tasse[1] and Karen Bradshaw[2]
Department of Computer Science
Rhodes University, P. O. Box 94, Grahamstown 6140
Tel: +27 46 6038291, Fax: +27 46 6361915
email: g09f5474@campus.ru.ac.za[1]; k.bradshaw@ru.ac.za[2]

***Abstract*—The advancements in virtualization technologies have provided a large spectrum of computational approaches. Dedicated computations can be run on private environments (Virtual Machines), created within the same computer. Through capable APIs, this functionality is leveraged for the service we wish to implement; a computer power service (CPS). We target peer-to-peer systems for this service, to exploit the potential of aggregating computing resources. The concept of a P2P network is mostly known for its expanded usage in distributed networks for sharing resources like content files or real-time data. This study adds computing power to the list of shared resources by describing a suitable service composition. Taking into account the dynamic nature of the platform, this CPS provision is achieved through an *on-the-fly* clustering algorithm. Moreover, this service would be highly beneficial in a corporate environment.**

***Index Terms*—Computing power service, virtualization, computer resources, P2P.**

I. Introduction

WEB technologies nowadays offer greater flexibility of service creation, from mere file transfers to clouds. In effect, we are witnessing more and more diversity of web services, and how they are increasingly shaping our computing experience. Cloud designs are becoming user-centred, thereby offering a ubiquitous web interaction. Though not entirely novel, *clouds* are emerging IT delivery models that bring a new insight on how to reason about computing services. To cite the definition in Stratus's white paper, "Cloud computing is the use of networked infrastructure software and capacity to provide resources to users in an on-demand environment" [1]. The term "resource" has also gained a wider meaning; it may stand for information, storage or even pure computing power. In this work, we focus on the latter, as an increase in computing speed is an undeniable advantage for user computations.

Resource-intensive applications are being developed both in research institutions and commercial industry, making computing power (notably processing cycles and memory) an important entity in the success of their operations [2]. Fortunately, the consolidation of virtualization techniques comes with benefits such as resource reservation and/or sharing for local (host) or remote (guest) use. The latter is of importance because it adds another dimension to how a resource can be utilized. Rather than having a resource idle because it is not being used locally, it can be made available to a remote user. The core of our intended service is to be able to serve, conveniently, the computer resource of a node to a different (remote) node. The fact that computing power is a low level entity and the operating environment consists of distinct nodes, requires transparency and reliability, respectively. In doing so, we comply with Verizon's definition that a computing service is an on-demand computing platform suitably set up to satisfy user needs and to exhibit ease of use [3].

The authors would like to acknowledge the financial support of Telkom SA, Stortech, Tellabs, Eastel, Bright Ideas 39 and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

The backbone of our computing power service (CPS) structure is built using P2P systems, where a node may have resources to share with its peers. This sharing of computer resources by leveraging processing cycles, cache storage, and memory is referred to as P2P computing [2]. We assume that peer nodes are not permanently busy, or that there exists nodes fully dedicated to sharing. Therefore, we stipulate that such nodes should *work* during their "idle" times. Effectively, one can maximize resource utilization within a P2P system by minimizing starvation. This suggests a P2P cluster implementation, where a node may alternate between being a worker and a client.

This paper is organized as follows. Section II presents a background study on the computing trends introduced here. From this, we identify in Section III the issues related to those trends, together with ones that arise from our service. In Sections IV and V, respectively, the approach to solving the identified issues and a successful design of the intended approach are presented. In, the last two sections (VI, VII), we propose potential applications of the work presented and our conclusions.

II. Background and Related Work

The CPS presented in this paper makes use of Virtual Machines (VMs) and related facilities to provide suitable computing-intensive workstations, while maintaining transparency and reliability in the system. Based on this statement, it is clear that one concept is key to the success of this research: virtualization. For a good grasp of the problem at hand, a background study is performed on the latter, together with related implementations.

A. Virtualization

1) Overview: In a virtualized environment, computing environments can be dynamically created, expanded, shrunk or moved. Virtualization refers to the abstraction of logical resources away from their underlying physical resources [4]. It is a key element for cloud computing because it provides important advantages in managing resources and thus enhancing business value [1]. A virtualized environment

not only makes the underlying platform transparent to the user, but it is also an efficient solution for computer resource sharing and scheduling. Virtualization is the best way to provide fundamental resource abstraction. Live migration is possible, avoiding an interruption of ongoing activities, critical for real-time computing. It also permits multiples and distinct Operating Systems to safely coexist on the same platform, thereby enhancing productivity [5].

2) Libvirt: Practically, an external resource consumer is a VM, commonly referred to as a guest machine. Libvirt facilitates VM instantiation by providing resource management and monitoring library calls [6]. It is a rich library for interacting with virtualization hypervisors. Its API allows a variety of application designs by supporting different types of hypervisors (see Fig. 1). The most popular is QEMU, a generic and open source machine emulator for full virtualization. It supports Xen and KVM hypervisors.

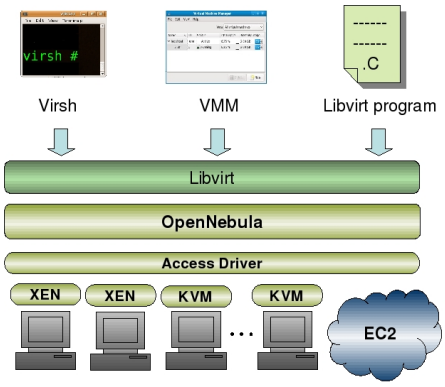

Figure 1: Libvirt architecture.

Several researches have been carried out by taking advantage of virtualization to provide a user-aware computational environment.

3) Infrastructure as a Service: There are systems that provide computer infrastructures by allowing users to run and control VM instances deployed on them. OpenNebula [7] and Eucalyptus [8] are both mature open source projects implementing IaaS. Both, in their distinctive way, provide an ecosystem of tools that can transform an existing computer infrastructure into a cloud. Physical resources of such an infrastructure are managed through a virtualization platform. Fig. 1 depicts the position of OpenNebula in this platform. These projects enable an enthusiastic user to set up his own computing environment such as grid interfaces for batch jobs, or dedicated servers for web servers. Such an environment will probably need a scheduler with advanced reservation and resource pre-emption features. Haizea is a resource lease manager, through which leases can be requested either as-soon-as-possible (best-effort) or in reservation [9].

With these solutions, the clustering of resources is automated and the obtained cluster self-managed. However, the process of consuming these resources is left to the users; i.e., VM set-up is not autonomous. To resolve this, a study on providing computing services is needed.

B. Computing Services

1) Computing Trends: The concept of computing services has evolved over time with respect to the following trends:

- Mainframe computing: many people share a single computer (a supercomputer). Because mainframes are cumbersome and only available to a handful of individuals, this form of computing is omitted in our study.
- Desktop computing: one person, one computer. This is the most popular form of computing at the moment, because of the incredible number of personal computers (workstations).
- Ubiquitous computing: many computers serve one person. This is the future we aspire to, where a single user can ubiquitously access multiple services or resources.

These trends are described and compared thoroughly in [10], where the author claims that ubiquitous computing will become more dominant as we observe an exponential increase in number of computers over time (See Fig. 2).

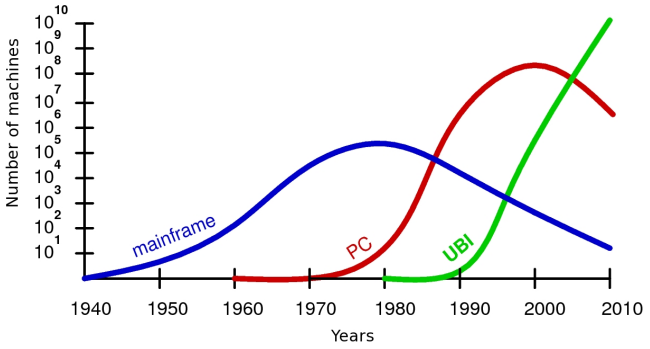

Figure 2: The major trends in computing [10].

2) Service Composition: The ubiquitous computing trend suggests a dynamic interoperation of distributed services, via automated service provision or runtime creation of new applications. Given a service request, the runtime environment in response has to create a self-compliant solution deployment plan [11]. Of course, this dynamic service composition will depend on the selection of service providers with respect to parameters like availability, performance, load, monetary costs or quality of service (QoS). The existing service composition of P2P systems implements this, but struggles to recompute a deployment plan during actual runtime execution. In [12], Prinz et al. proposed a service composition framework (SCF), enforcing each sub-service to implement four methods that will ensure their execution within the framework: one to execute the service on passing parameters, a second to stop it, a third for checkpointing the service current state and a fourth to return the state. Peers that participate in the SCF network provide services satisfying these properties. The SCF network enforces robustness during runtime execution using Logical Service Groups (LSGs). An LSG contains a set of available peers that locally provide a dedicated service. During the service execution, one peer of the LSG executes the service and *n* other group members monitor its heartbeat (see Fig. 3). Failures within the network are compensated by migrating the composite service; i.e., creating and applying a new deployment plan.

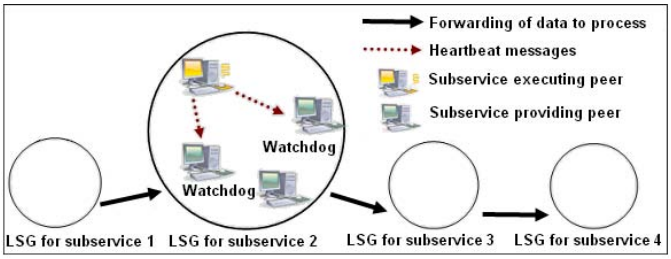

Figure 3: Service Composition based on LSGs.

In brief, [11] and [12] introduce an optimized service creation and execution of composite systems: i.e., *context aware computing*.

3) Example of CPS: Due to cost considerations, small and medium-sized enterprises (SME) generally favour using P2P computing. The Narada Brokering CPS (NB-CPS) combines computational grids, distributed objects and P2P networks under a hybrid environment [2]. NB-CPS argues to solve issues such as system resilience, fault tolerance, efficiency of job scheduling, and instability in congested network environments. Rather than using VMs, it provides a JMS (Java Messaging Service) API for developers to create distributed applications. Additionally, it establishes a framework for communication through a message-oriented middleware. For scaling purposes, its messaging infrastructure is based on a distributed broker network, where a broker is part of a cluster that is part of a super-cluster. It is basically a Distributed Hash Table (DHT) for decentralized information management.

However, this attempt only partially solves the problem, since it is only suitable for developers.

III. Problem Definition

Before a solution attempt is made, we provide an explanatory presentation of the problem at hand in this section.

A service in this paper's context is a computational response to a task description, based on a running VM. A peer node can either be a client (requester) or a worker node (provider). The set of worker nodes forms the computing unit. A client benefits from the computational capabilities of a worker node through the dynamic service creation, provided that the worker can host related VM(s). This primarily depends on the node's self-specified restrictions on its resources (e.g., amount of shareable memory), established beforehand (i.e., prior to joining the system).

From the previous section, it is clear that our basic tool for creating a CPS is virtualization. Evidently, the first step to guarantee hosting of any CPS, is to perform resource reservation and isolation on worker nodes. Virtualization appears to be the native solution for this. However, there is still a long road between it and a ready-to-use cloud service, considering especially the attributes of a P2P environment. We start the journey by explaining the key requirements.

Transparency: We have already mentioned the importance of seamless service creation and convenient service interaction. The service should be easily accessible ("point and click" service access). Through the term transparency, we emphasize the fact that when a user runs a task remotely through our service provision system, s/he should be comfortable doing so. We identify two steps here.

1) Find a suitable host for the intended service (cloud/grid computing).
2) Provide a workspace abstraction for the user (ubiquitous computing).

Reliability: Although issues like bottlenecks or single points of failure are not of great concern in P2P systems, the permanent node churn induces a notable risk for hosting any external computation. The way this risk is dealt with defines the reliability of a service, and the system as a whole. We identify two related factors.

- Fault tolerance: accidental failures may occur on a host.
- *Volunteering* based networking: nodes can join or leave the system freely.

With respect to the targeted environment and the criteria (transparency, reliability) just presented, we complement the definition of CPS provision.

IV. Proposed Solution: on-the-fly Provisioning of CPS

A new cloud model can be applied to P2P networks, to provide on-the-fly cloud solutions. In response to this statement, we propose in this section an on-demand service provision system.

At this point, we understand there are issues with hosting a service on a peer node (the worker) and issues with making it available to another peer node (the client). Therefore, we identified two complementary views: the client view and the worker view. To solve the problem, we take into account both views.

A. User-side: Request Processing

We aim to solve identified user-side issues (notably transparency) particularly by processing requests. First, all user interaction is done via a web browser, for compatibility and portability reasons. User requests are captured through this interface and processed accordingly. As depicted in Fig. 4, we need to transform a submitted user task description into a VM description, then start the VM on a node (thereby executing the task), and give access (control) of this VM back to the user.

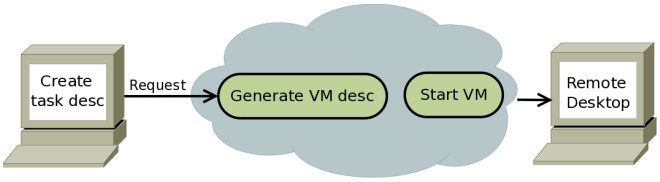

Figure 4: Request work-flow.

To guarantee a familiar user interaction, a remote desktop technology is employed. Instead of creating a new user interface for VMs, we use already available and well designed GUIs, thereby providing a perfect desktop integration. Effectively, part of the service is to provide a web server, with a remote desktop protocol capability.

B. Host-side: Response Handling

On the host-side, we have identified issues with setting up a host-ready environment for a CPS. Native P2P networking solutions are used to solve these issues; a best-effort algorithm for instance. We start by hypothesising that each peer node has equal qualitative potential. That is, the only difference between the nodes with respect to this system is their computing power.

Response handling is done in two phases: allocate the VM job to a node to start its execution, and monitor this node for failure handling. These phases are achieved, respectively, by using the following methods:

- On-the-fly node detection: We recognize that a subscription-based method, using DHTs (Distributed Hash Tables), as described in Section II, has great success. However, we assume a small to medium scale

P2P network, where a DHT is unnecessary, and its overhead cannot be tolerated. But a fully distributed node discovery model is still needed. So, we use a reactive protocol that locates suitable nodes when there is a pending task. This is based on multicasting, whereby capable nodes respond to the multicast task request.

- Failure handling: When a peer node has an overload or is not able to keep on hosting running services (VMs), the extra load should be migrated to different nodes; a load sharing mechanism is implemented to achieve this. In the case of complete failure, the task is resubmitted.

We have discussed the main issues that need to be resolved before progressing in our work. The next section complements the solution presented here by presenting a prototype design and implementation.

V. Design and Implementation

Cloud services can be automated in a hostile environment; i.e., integrate cloud solutions transparently onto a running computing network. The CPS, the cloud service being offered, is a ready to use service that responds to a user's computing needs. This section combines the ideas proposed in the previous section with specific design and implementation insights and approaches.

The provisioning model has several components, explained in detail in Section V-B. Further, we show how to create a service on a particular node, how it is instantiated and made accessible for use.

A. Resource Reservation and Allocation

As hypothesised, CPU cycles, memory and network bandwidth (hardware resources) are the targeted resources; the computing power needed for external usage. It should be noted that storage is also a resource, even though it will not be persistent, as it would be ordinarily. The external user (client) who benefits from the *shares* of a resource provider is the resource consumer; though practically, it will be the client VM. There is need for a management module to bridge these two entities; i.e., reserve resources from the provider and allocate them, as requested, to the consumer.

```
<domain type='qemu' id='2'>
 <name>username1</name>
 <memory>256144</memory>
 <vcpu>1</vcpu>
 <os>
  <type arch='x86_64' machine='pc-0.12'>hvm</type>
  <boot dev='cdrom'/>
 </os>
 <devices>
  <emulator>/usr/bin/qemu-system-x86_64</emulator>
  <disk type='file' device='cdrom'>
   <driver name='qemu'/>
   <source file='/home/ubuntu-8.04-desktop-i386.iso'/>
   <target dev='hdc' bus='ide'/>
   <readonly/>
  </disk>
  <video>
   <model type='cirrus' vram='9216' heads='1'/>
  </video>
 </devices>
</domain>
```

Figure 5: Libvirt XML description of a VM for the QEMU emulator.

Fortunately, the implementation of this module is simplified by the Libvirt API [6]. One of the facilities of the API is to parse a VM description, constructed with a well documented XML schema. This allows us to specify how shared resources are allocated among hosted VMs. Fig. 5 is a sample snippet of the XML description that can be used to create a VM. Once this description is passed to the API, through the *virDomainCreate* call, it takes care of the rest; i.e., actual allocation of resources.

B. Service Provision Model

As explained in Section III, to transform a desktop platform to a cloud platform, several modules need to be developed. Contributing nodes may have different potential and will have different load cycles. Each node is only liable for its own execution and may not be aware of the existence of others, unless absolutely necessary (see Section V-C2). The outcome is to have each node operate independently, automating its service provision to satisfy both host and guest(s) computations. No latency should appear on the host, as it accepts a guest's job only if it is capable of completing it.

Here we describe the necessary components needed on a node for the service provisioning back-end. Each prospective resource provider implements a stack of modules, which will elevate it from a resource provider state to that of service provider (see Fig. 6).

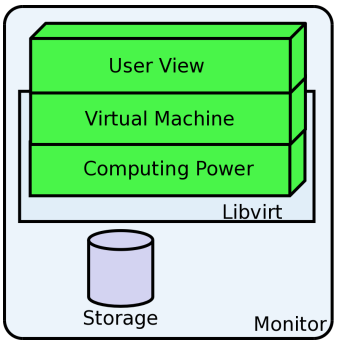

Figure 6: Service provision components.

1) User view layer: This is where the desktop abstraction is done. We stipulated that a familiar user interaction is needed, and that *remote desktop* is the technique adopted. Effectively, this layer implements a remote desktop protocol. Each VM is attached with a VNC (Virtual Network Computing) server, exporting its desktop. Therefore, users can remotely control or monitor their jobs, running through VM(s) as if they were running locally. As mentioned earlier, any user-to-service interaction will be done through HTTP. So, we use an HTTP proxy for VNC based on JavaScript, called noVNC [13].

Though we chose, for user comfort, a remote desktop interface, a text based (serial) interface is also possible. The latter is obviously less bandwidth expensive, but may only suit technical users.

2) Virtual machine and computing power layers: These layers operate as described in the previous section (resource reservation). The creation and monitoring of a VM is achieved with Libvirt, using the description submitted.

3) Storage layer: A temporary storage is created for convenience. Evidently, a storage unit is needed to accompany the computing service. This will usually be an empty disk, but can be a *bootstrap*, depending on the user's request. It has a virtual file system, created through disk imaging. To

perform this, we used the *guestfs* library in [14], though other methods also exist. Once a VM is terminated, its disk image is discarded; the user has of course a choice to retrieve a copy (compressed) thereof.

4) Monitor layer (Watchdog): Because nodes can join and leave the network freely, availability of a started service is the system's responsibility. The role of this layer is to oversee a service instance execution and trigger events accordingly. It basically makes sure that the different components start, run and terminate successfully. In case of complete failure, which means failure of the monitor itself, the user should resubmit the task.

Caching protocols, are also implemented using the *checkpointing or suspend* facility available in the Libvirt API.

C. Architecture

Here, we complete the model in Section V-B by describing the full architecture of the P2P CPS. Fig. 7 depicts the logic of the overall system.

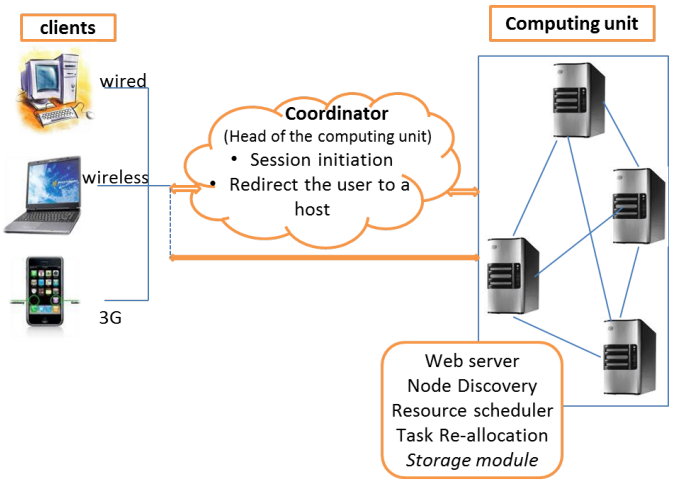

Figure 7: System architecture.

The web server is the system's entry for external users; they construct and submit their task here. It is part of the coordinator, a middleware unit that responds to user requests and sets up the interaction between the user and the corresponding service. It hides the complexity of the computing unit to offer an explicit view for the client. Practically, the coordinator is addressed through a DNS name that holds a variable value. This DNS name maps to the address of the current head node of the computing unit. The latter is chosen mostly based on its uptime value. DNS update packets are used by a peer node to claim the head node position.

For a user to access its service, a flow of events are spawned from its request. The following describes the milestones of a service instance.

1) Service Discovery and Service Instantiation: The service discovery is mostly based on a host discovery algorithm. The latter suggests a host capable of handling a giving request. During this phase, a node with the minimum computing power requirements is detected among the pool of connected nodes. This node will become a host by implementing an adequate service, with respect to the model explained in Section V-B.

As suggested in Section IV, a multicasting based protocol is used to find a node that can attend to the client. Peer nodes within the system belong to a particular multicast group, defined by a multicast address. So, a node communicates with its peers by sending event messages to the predefined multicast address; this process is called event publishing. With respect to this, we define a protocol called Publishing Event Protocol (PEP).

When the coordinator receives a client request, it generates a corresponding PEP packet, which is multicasted across the system. Fig. 8 is a self explanatory representation of a PEP packet.

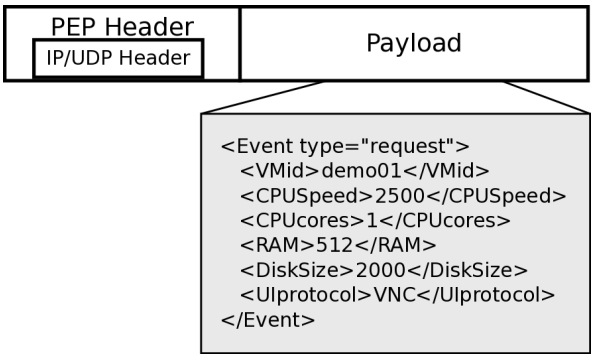

Figure 8: A sample PEP packet.

Only capable hosts respond to this packet. They simply acknowledge it using an ACK packet; this process is called event subscription. The coordinator captures the addresses of all these nodes, and chooses the prospective service host to be the node that subscribed first. Then, it acknowledges back to this node, adding the addresses of other subscribers. These addresses could be useful to the host, when a service migration is needed.

After the publish-subscribe events, the service instantiation phase follows. It uses the task's description acquired from the PEP packet. As shown in Fig. 8, this packet elaborates only the computing power requirements. To these requirements are added the user-view functionalities (see Section V-B1). Additionally, the user may submit complementary files to automate the VM initialization and have it ready for use when it is started.

2) Service Deployment and Maintenance: This summarises three steps: starting, scheduling, and re-allocation of a service instance. During the deployment phase, we ensure that the service has started correctly and the user has access to her/his VM. In case this fails, the coordinator re-deploys the service on another node. It finds the latter using a *best-effort* scheduling algorithm. This algorithm merely looks for an available node satisfying the minimum computing power requirements.

The service maintenance module implements the fault tolerance. When a fault occurs on a host, its tasks are re-allocated, based on the scheduling algorithm just introduced. A fault will either be a host decision to stop external jobs, or its limitation to pursue their execution. To detect faults, the module interacts with the monitor layer described in Section V-B4. Its core implementation uses virtualization capabilities such as suspend/resume and migrate, available in Libvirt.

VI. Applications

Our research aims to build a uniform and self-managed cloud infrastructure by clustering natively independent and heterogeneous computer resources, to provide a uniform and

robust resource. The advantages of having more resources include the potential for parallelism, multiple points of failure, and scalability. With respect to this, we have identified a number of environments where the research deliverable is applicable.

Target environments

Because of the nature of the CPS proposed in this paper, we believe it would be most suitable for a corporate environment where computer resources are not necessarily centralised and are underutilized. Computing tasks should be assigned fairly among these resources. The potential to aggregate distributed resources in various enterprise collaborating functions is immense. In such an area, the system maximizes the use of resources through an intelligent, self-automated algorithm. Hardware resources can be donated without any physical displacement thereof. We describe two potential test environments.

- Research environment: a clustering of several labs (network of workstations) can outperform a supercomputer. Moreover, resources can be reserved beforehand for a dedicated batch computation (advance reservation).
- Real-time environment: consider a hospital environment. One can use other under-utilized resources to execute a real-time computation such as rendering a 3D image from a X-Ray scanner. Starvation should be minimized as much as possible; for example, if an image is currently being processed, run the next one on a different computer.

VII. Conclusions and Future Work

The paper proposed a novel method for the creation of cloud services. To facilitate and enhance users' computer experiences, a cloud is built within a P2P network. Peers form a cluster of potentially powerful computers (workers) on which external computations are virtually executed. The cloud merely provides a CPS system, adequate for the environment. It has built-in intelligence, which enables it to have the dual property of unity (from a user point of view) and component independence (from a peer point of view). Peer nodes contain a set of modules that enable them to be considered as a unit, a computing unit; this is an emergent behaviour.

Our solution works for a corporate environment, where scalability is not a huge factor, and networking configuration can be controlled. With respect to this, we doubt that the method for discovering nodes, which depends on multicasting, will work as desired in a different completely hostile environment. It is possible that multicasting may not be allowed in some networks, or may lead to huge latency due to the scale in which it operates. In this case, a different discovery model is required.

Additionally, rigorous experimental testing is needed to complement this research. We are currently working on this.

References

[1] "Server virtualization and cloud computing: Four hidden impacts on uptime and availability," White paper, Stratus Technologies, June 2009.

[2] M.-J. Tsai and Y.-K. Hung, "Distributed computing power service coordination based on peer-to-peer grids architecture," *Expert Syst. Appl.*, vol. 36, no. 2, pp. 3101–3118, 2009.

[3] Computing as a service (caas). Verizon. [Online]. Available: http://www.verizonbusiness.com/Products/it/cloud-it/caas/

[4] *Solution Edition for Cloud Computing*. IBMCorporation, 2009. [Online]. Available: ibm.com/systems/z/solutions/editions/cloud/index.html/

[5] A. Gavrilovska, S. Kumar, H. Raj, K. Schwan, V. Gupta, R. Nathuji, R. Niranjan, A. Ranadive, and P. Saraiya, "Abstract high-performance hypervisor architectures: Virtualization in hpc systems," March 2007.

[6] Libvirt: The virtualization api. [Online]. Available: http://www.libvirt.org

[7] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, pp. 14–22, September 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1638588.1638692

[8] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131. [Online]. Available: http://dx.doi.org/10.1109/CCGRID.2009.93

[9] B. Sotomayor, K. Keahey, and I. Foster, "Combining batch execution and leasing using virtual machines," in *Proceedings of the 17th international symposium on High performance distributed computing*, ser. HPDC '08. New York, NY, USA: ACM, 2008, pp. 87–96. [Online]. Available: http://doi.acm.org/10.1145/1383422.1383434

[10] M. Weiser. (1996) Ubiquitous computing. [Online]. Available: http://www.ubiq.com/hypertext/weiser/UbiHome.html

[11] A. V. Konstantinou, T. Eilam, M. Kalantar, A. A. Totok, W. Arnold, and E. Snible, "An architecture for virtual solution composition and deployment in infrastructure clouds," in *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, ser. VTDC '09. New York, NY, USA: ACM, 2009, pp. 9–18. [Online]. Available: http://doi.acm.org/10.1145/1555336.1555339

[12] V. Prinz, F. Fuchs, P. Ruppel, C. Gerdes, and A. Southall, "Adaptive and fault-tolerant service composition in peer-to-peer systems," in *Proceedings of the 8th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*, ser. DAIS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 30–43. [Online]. Available: http://portal.acm.org/citation.cfm?id=1789074.1789078

[13] J. Martin. Vnc client using html5 (web sockets, canvas) with encryption (wss://) support. Sentry Data Systems. [Online]. Available: https://github.com/kanaka/noVNC

[14] (2009) Library for accessing and modifying virtual machine images. Libguestfs. [Online]. Available: http://http://www.libguestfs.org/

Mr. Ghislain Fouodji Tasse completed his Honours in Computer Science in 2009 at Rhodes University. Ghislain is now in his second year of MSc in Computer Science, funded by the Centre of Excellence (CoE) at Rhodes University.

Dr. Karen Bradshaw is a Senior Lecturer in the Computer Science Department at Rhodes University. Her research interests are distributed computing and robotics frameworks.