A Virtual Cluster Manager using a Hierarchical Management Model for Cloud Infrastructure

Pongsakorn U-chupala*, Kohei Ichikawa†, Hirotake Abe, Susumu Date and Shinji Shimojo‡
*Department of Computer Engineering, Kasetsart University, Bangkok, Thailand
Email: b5105274@ku.ac.th
†Central Office for Information Infrastructure, Osaka University, Osaka, Japan
Email: ichikawa@cmc.osaka-u.ac.jp
‡Cybermedia Center, Osaka University, Osaka, Japan
Email: {habe, date, shimojo}@cmc.osaka-u.ac.jp

***Abstract*—With cloud computing technology, specifically infrastructure as a service (IaaS), it is possible to conveniently deploy a virtual machine on the cloud. There are also a lot of cloud toolkits available to help administrators setup IaaS cloud. However, to deploy a set of virtual machines and configure them to work together as a virtual cluster is still a complicated and tedious task. Most cloud toolkits offer not comprehensive virtual cluster management solution but individual virtual machine management. To address this problem, we propose virtual cluster manager, a unified utility for virtual cluster management. This paper describes a design and implementation of virtual cluster manager built on top of existing cloud toolkit, OpenNebula, as well as the case study of this utility through the comparison with virtual cluster deployment with traditional methods.**

I. Introduction

With cloud computing [1] being a buzzword recently, there are a wide range of cloud computing toolkits available nowadays. Each provides services in different abstraction. In this research, we focus on Infrastructure as a Service (IaaS) [2] paradigm. These toolkits offer the ability to manage resources ubiquitously with united cloud interface.

Even though virtual infrastructure (VI) provides virtually limitless computation capability to user, in reality, performance of each virtual machine (VM) running on the infrastructure is still limited by physical host machine. It turned out that we have already solved this problem with cluster computing. We could use this same methodology and create a cluster of VM instead of physical machine. This technology is called Virtual Cluster (VC) [3].

However, most VI toolkits (such as OpenNebula [4] and Eucalyptus [5]) only focus on individual VM management. Although it allows us to build VC with these existing toolkits by configuring and deploying each VM manually, there is no convenient way to manage VC as a whole yet. To simplify this process, we propose a utility that manages VC with existing VI toolkit. This utility will provide users another layer of abstraction that allows users to view the system as a repository for VC.

In this research, we have developed CLI Utility called *onevc* for OpenNebula 3.0 (VI toolkit) which provides the ability to manage VC as a whole. This utility works alongside other built-in CLI utilities seamlessly and provides interface similar to existing built-in utilities for consistency.

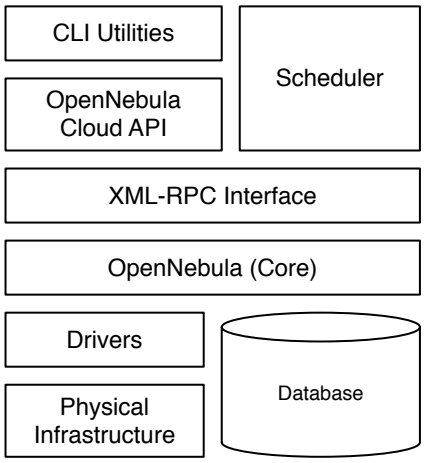

Fig. 1: OpenNebula software stack (simplified) [6].

This paper is structured as follows. Section II describes OpenNebula in general, similar utility called “oneservice” and differences between “oneservice” and this research. Section III describes the architecture of VC manager. Section IV describes VC manager implementation based on OpenNebula VI toolkit. Section V demonstrates cases where VC manager performs better than existing utilities and their measurement. Section VI is the conclusion and future directions.

II. OpenNebula as a Building Block

A. OpenNebula

In this paper, we refer to OpenNebula. Thus, it is needed to briefly explain about OpenNebula and its architecture for later discussion.

OpenNebula is a VI manager. It orchestrates VM placement, networking and resources such as disk images and exposes unified cloud interface to its users, thus creates an IaaS cloud. It enables creation of private cloud with existing resources. OpenNebula also is characterized with its modular architecture, which allows it to work with several hypervisors. This

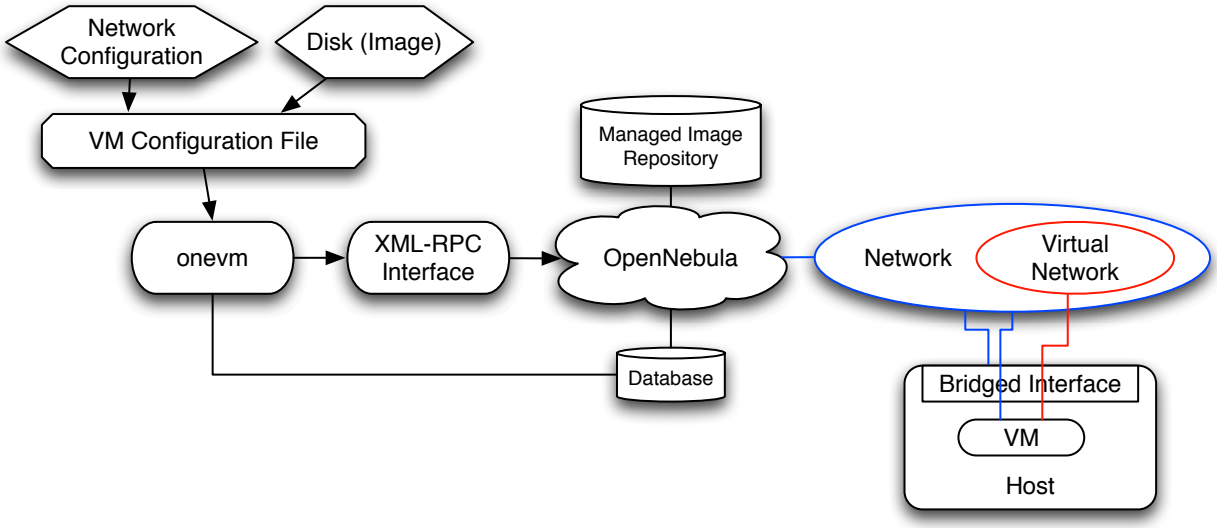

Fig. 2: onevm VM deployment process with one deployed VM.

feature also enables OpenNebula to support creation of hybrid cloud as well.

To archive a high level of scalability, OpenNebula architecture is designed to be modular. Using drivers, OpenNebula can be plugged into a wide range of datacenter services. The software stack is also divided into multiple layers. This approach allows us to modify and extend OpenNebula as needed. Figure 1 show the architecture of OpenNebula software stack.

To manage the cloud, OpenNebula is shipped with a number of built-in CLI utilities such as *onevm*, *onevnet*, *onehost* and *onetemplate*. These utilities provide the information of the cloud as well as perform various tasks such as VM deployment and deletion on each subsystem of OpenNebula. Each utility connects to OpenNebula via OpenNebula Cloud API (OCA) and provides abilities to manage the cloud. OpenNebula maintains pools of hosts, disk images, network configurations, VM templates and VM instances with *onehost*, *oneimage*, *onevnet*, *onetemplate* and *onevm* respectively. To register a configuration into each pool, users need to provide a template for each utility accordingly (i.e. supply network configuration template to *onevnet* to add network configuration the pool) with the exception of *onehost* which accepts parameters directly from the shell. VM template could then refer registered resources (network configurations, disk images, etc.) to be used.

After all configurations are in place, to deploy VM, users need to register VM templates to the template pool and use *onetemplate* utility to instantiate VM instances or just supply the templates to *onevm* utility directly. Figure 2 shows deployment operation flow of *onevm*.

B. oneservice

There is a concept of service which is consisted of multiple VMs working together being developed for OpenNebula. For an example a web server service could consists of web servers, database servers and load-balancers. oneservice [7] utility allow user to describe service and its dependencies with

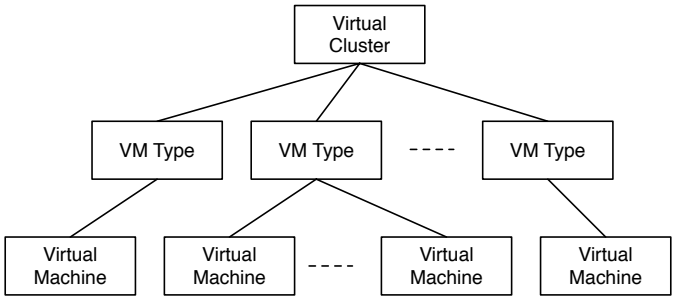

Fig. 3: Hierarchical model of virtual cluster.

Service Description Language (SDL). This idea is similar to the virtual cluster but the key difference is service management allows user to manage as group of virtual machines as a whole while virtual cluster management allows user to manage the whole cluster or just a subset of machines in the cluster.

III. Architecture

Similar to how *onevm* utility of OpenNebula works, to describe a VC to VC manager, we have adopted hierarchical template system. This template system allows user to describe VC attributes in one single template file. This template extends existing VM template system for VC deployment. VC manager takes VC template as an input and then registers the VC to the database. After that, users can instruct the manager to do management operations on registered VC or each VM type individually. Supported management operations are create, deploy, suspend, stop, resume, delete, list and show.

A. Virtual Cluster Model

In this research, we have defined VC with a hierarchical model. A VC consists of one or multiple VM type, served as a grouping method for VM. A number of VMs could be

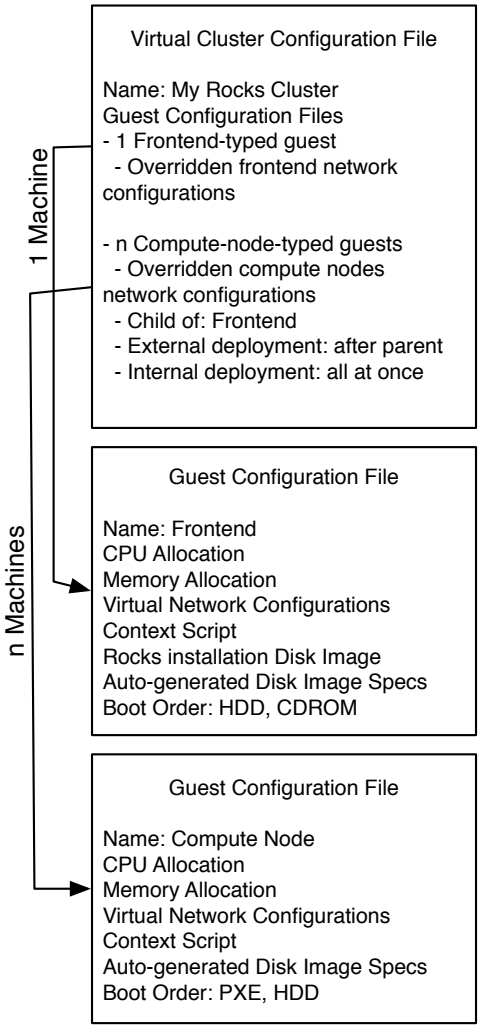

Fig. 4: Example VC template (detail omitted).

instantiated in each VM type. Figure 3 shows hierarchy of VC in this model.

VM types in VC also forms its own hierarchy. This hierarchy is used along with other configurations to determine the deployment order of each VM in a VC.

One example is a VC with one frontend machine and a number of compute nodes. To represent this VC with this model, 2 VM types to represent frontend VC and compute node VM are used. The frontend VM type contains only one VM while the compute nodes VM type contains *n* number of VM with the compute nodes VM type being a child of the frontend VM type.

B. Virtual Cluster Template

To describe a VC with our model, we use template. For each VC, a VC template is used to describe its properties. This VC template syntax is the same as OpenNebula template syntax to preserve consistency with others built-in utilities, to be more familiar to user. VC properties may include information such as name, VC types and number of VM to deploy for each VM type.

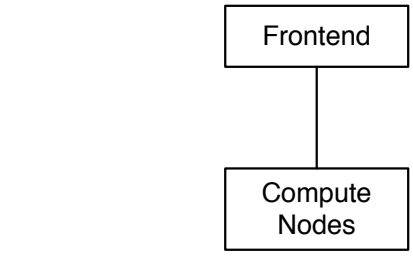

Fig. 5: VM type hierarchy of example VC template.

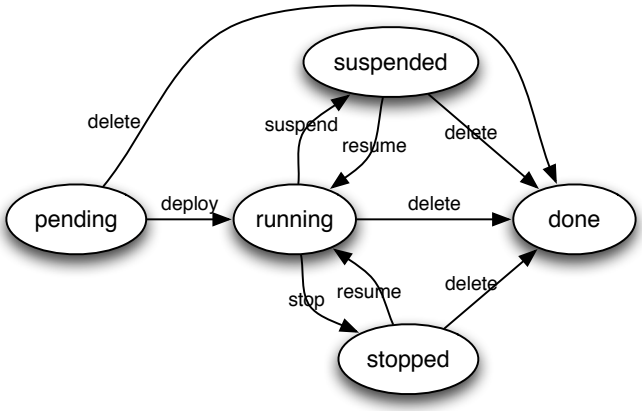

Fig. 6: VC and VM Type state diagram.

To describe each VM type in VC template, we use the same OpenNebula VM template structure. We also allow pointing to external VM template path or template id of template registered in *onetemplate* built-in utility. If VC template is defined with either of these 2 methods, we also allow VM type template overriding from VC template as well. This allows VM type sharing between many VC templates as we could modify VC specific configuration such as networking as needed.

Each VM type is also configurable with external deployment policy such as “with parent”, “after parent” or “manual” and internal deployment policy such as “all together” or “one by one”. These policies will be used to determine deployment order of each VM for automatic cluster deployment. Figure 4 is shown as an example of cluster VC template.

To determine deployment order, firstly, we use external deployment policy and VM type hierarchy to determine deployment order of VM type. VM type that does not have parent will be deployed first. If there are any child VM type of these VM which external deployment policy set to “with parent”, those VM type will be deployed at the same time. After all of these VM types successfully deployed (reached “running” state), their children will be deployed next, and so on until all VM type is deployed. For deployment order of each machine in the same VM type, we use internal deployment policy.

C. Virtual Cluster State

We assign state to each VM type in a VC and the VC itself in order to keep tracking. During initialization, all VM

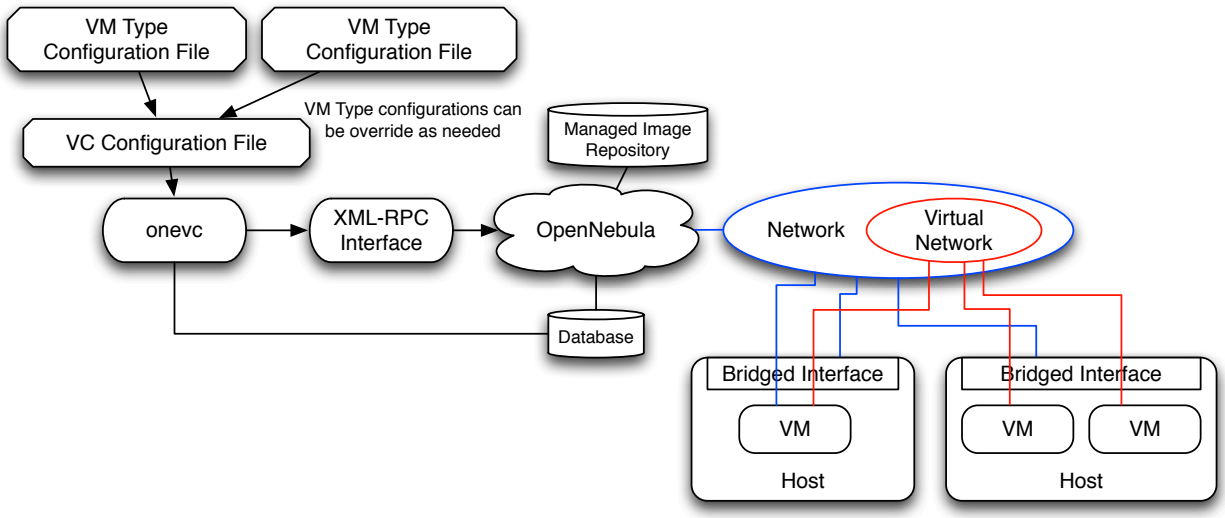

Fig. 7: onevc VC deployment process with 1 deployed VC of 3 VM.

type will be assigned to "pending" state. State of VM type is shifted along with VC management operations (deploy, suspend, resume, etc.). State shifting is described in Figure 6. These states are designed to support OpenNebula's VM state [8] and bear a similarity with it.

When all VM type in a VC is in the same state, VC itself is assigned with that state as well. Otherwise, the VC is assigned "shift" intermediate state to indicate that the state is being shifted and currently inconsistent.

IV. Implementation

VC Manager has been implemented as OpenNebula CLI utility and named *onevc* for consistency with other OpenNebula CLI utilities. *onevc* is written in Ruby just like other built-in CLI utilities. It works along with other built-in CLI utilities on the same layer and relied on the same OCA interface. However, we need to create some more tables in the database to hold VC information.

A. Database

We use existing mechanism provided by OpenNebula to manage VM so it uses existing `vm_pool` table as well. For VM type, we use existing template system and by storing VM type configuration as template in existing `template_pool`. We define 3 additional tables in the database, `vc_pool`, `vc_node` and `vm_type`. `vc_pool` stores VCs registered to *onevc*, their configuration from their template and their state. `vm_type` associates each VM type specified in VC template with the actual VM type configuration stored in `template_pool`. `vc_node` associates each VM with its VC in `vc_pool` and VM type in `template_pool` as well as recording its state. Figure 8 shows the relationship between each table.

B. Operation

User can supply VC template to *onevc* in the same fashion as with *onevm* utility. *onevc* validates the template, puts it into *onevc* `vc_pool` table and assign all VM type and the VC itself with "pending" state. After that, *onevc* creates all VC with OpenNebula then puts all of them in OpenNebula's "pending" state, waiting for deployment. Users could then deploy registered VC and do management tasks with *onevc* utility. Figure 7 shows the VC deployment process of *onevc*.

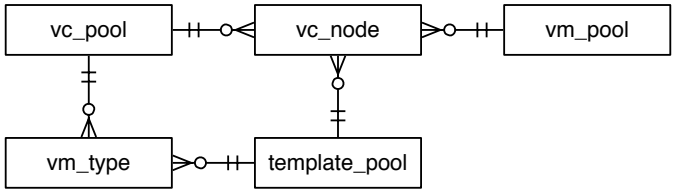

Fig. 8: ER diagram of onevc database.

onevc utility support the following operations.

- `create`
  register VC template to *onevc* utility and initialize VC
- `deploy`
  deploy specified VC or specified VM type according to VM type hierarchy and deployment policy in VC template
- `suspend`
  suspend VC or specified VM type in reverse order of deployment
- `stop`
  stop VC or specified VM type in reverse order of deployment
- `resume`
  resume VC or specified VM type in the same order as deployment
- `delete`
  delete VC or specified VM type in reverse order of deployment
- `list`

list registered VCs and their statuses

- `show`
  show specified VC information

V. Case Study

To show advantages of our VC manager design, the following cases are considered.

A. Ease of use

We use number of commands used to deploy a VC to demonstrate ease of use of the manager. The less command needed, the more convenient it is. Using the example VC template in Figure 4, we deploy the VC with existing utility and *onevc* utility.

1) With existing built-in utility: Firstly, we created 2 templates for frontend and compute nodes and register it to *onetemplate* built-in utility with this command.

```
$ onetemplate create
<path_to_templatefile>
```

Then, we instantiate 1 frontend VM followed by a number of compute nodes. To instantiate VM with *onetemplate*, we used this command.

```
$ onetemplate instantiate <template_id>
```

Where `<template_id>` is template's id given to registered template. It took two commands to register templates, one command to instantiate frontend and the other to instantiate each compute node, so we used 3+n commands in total. We also need to maintain each template individually and make sure that configurations in both templates worked together.

2) With onevc utility: Let the name of VC configuration file is `mycluster.one`, Firstly, user needs to register the template with this command

```
$ onevc create mycluster.one
```

After registration, user can use this command.

```
$ onevc deploy mycluster
```

According to deployment policies in the template, *onevm* will deploy frontend VM first. After it reach "running" state, it will deploy *n* number of compute nodes at once. User can choose to deploy each VM type manually by using these following commands.

```
$ onevc deploy mycluster frontend
$ onevc deploy mycluster computenodes
```

Note that if frontend VM type isn't at "running" state when user manually deploy compute nodes then *onevm* will try to move frontend to "running" state by deploying or resuming as necessary.

In total, we used 2 commands for automatic deployment and 3 commands for manual VM type deployment regardless of how many compute nodes are there in the cluster.

B. Template Reusability

With VM type hierarchical structure, it is possible to reuse VM type across multiple VC. To demonstrate this point, we use number of templates needed as a measurement. In this case, we will deploy 3 virtual Rocks Clusters [9] with different number of compute nodes for each VC (*l*, *m* and *n* compute nodes).

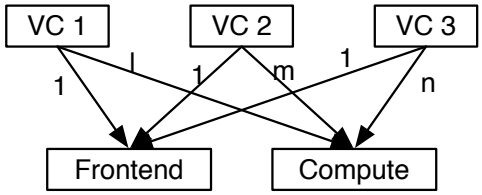

Fig. 9: VM type sharing between multiple VC.

1) With existing built-in utility: For each VC, we need 2 VM templates file for frontend node and compute nodes. Even if some part of frontend node and compute nodes share the same configuration, there are differences between each template (such as network configuration to use) so configuration file need to be duplicated and maintained individually. In total, 6 VM templates are needed. Each node also needs to be deployed manually, so OpenNebula cannot keep track of number of compute nodes for each VC at all. Figure 9 shows VC Hierarchy of these configurations.

2) With onevc utility: Because frontend node and compute nodes for each VC bare similarity, we define them with single VM type template for each one. Each VC can just point to these 2 VM type template, specify number of compute node needed and override unique configuration for each VC. We need 3 VC templates and 2 VM type templates so there are 6 templates needed in total.

VI. Conclusion and Future Work

In this paper, we have described a convenient utility for VC management with existing VI toolkit. Instead of managing each VM in a VC individually, this utility offers a method for managing VC as a whole. In order to flexibly describe VC, we use hierarchical VC model along with the template system. This template system also offers policy configurations to be used alongside VM type hierarchy to determine deployment order. With this architecture, we have implemented *onevc* utility for OpenNebula and showed how to deploy typical VC with this tool through the comparison with existing built-in utility. Our case study implies that this VC manager make VC management simpler and convieneint.

However, we only look into VC management and leave VC template management totally to users. For this issue, we could create another utility to manage VC template. This utility could work in the same manner as *onetemplate* built-in utility of OpenNebula manage VM template but with VC template instead.

Acknowledgment

This research is partly supported by the joint research project titled "Research on structuring method of multiple overlay networks leveraging NGN", between Osaka University and NTT Cyber Solutions Laboratories. Also, this research was partly supported by the Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Young Scientists (B) [No. 22700052, 23700058]. The authors would like to thank Shimojo Lab for providing research facilities,

FrontierLab@OsakaU program for the opportunity and Japan Student Services Organization (JASSO) for their financial support.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[2] A. Ranabahu and E. Maximilien, "A best practice model for cloud middleware systems," *Towards bes*, pp. 41 – 51, 2009.

[3] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang, "Virtual clusters for grid communities," vol. 1, pp. 513 – 520, 2006.

[4] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14 – 22, 2009.

[5] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," pp. 124 – 131, 2009.

[6] "Scalable architecture and apis 3.0," 11 2011. [Online]. Available: http://opennebula.org/documentation:rel3.0:introapis

[7] W. Iqbal. (2010, 11) Opennebula service manager. [Online]. Available: http://opennebula.org/software:ecosystem:oneservice

[8] Managing virtual machines 3.0. [Online]. Available: http://opennebula.org/documentation:rel3.0:vm_guide_2

[9] P. M. Papadopoulos, M. J. Katz, and G. Bruno, "Npaci rocks: tools and techniques for easily deploying manageable linux clusters," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 7-8, pp. 707–725, 2003. [Online]. Available: http://dx.doi.org/10.1002/cpe.722