Master's Thesis

ClusterNap: A Generic Power Controller Interface for Green Clusters and Clouds

(ClusterNap: グリーンコンピュータクラスタとクラウドのための汎用電力コントローラ)

Amgalan Ganbat (48-126415)
(アムガラン　ガンバト)

Supervisor:　Professor Taura Kenjiro (田浦　健次朗)

February 6, 2014

Department of Information and Communication Engineering
Graduate School of Information Science and Technology
The University of Tokyo

Abstract

In order to save the ever increasing operational cost of computer clusters or cloud infrastructures, continuous and active management of resources has become an indispensable part of IT infrastructure management. One of the most promising approaches to reduce these costs is to use only the necessary amount of resources and switching the idle ones into a low cost/power consuming state. Complex dependencies among various types of resources, however, make it challenging to select the right resources to turn ON and OFF. We propose a generic, resource dependency aware power controller tool, ClusterNap, which users can use as a controller module and interface for their energy/cost aware systems and applications.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Until recently, resource management of IT systems such as computer clusters and datacenters used to imply mainly about workload scheduling and resource monitoring. However, since performance and scale of such facilities have increased tremendously in the last two decades, controlling the resources dynamically as well as interactively tends to become an indispensable part of resource management. In the following subsections, we explain why such a dynamic resource controlling is crucial and what are the challenges which motivated our work. The contribution of our work in explained as well. In this paper by *resource* we refer to either physical resources (i.e., servers, switches,or RAIDs), virtual resources (i.e., virtual machines, cloud instances), or certain services. In addition, we call each single entity of these resources a *node*.

1.1.1 The power consumption issue and energy-proportionality

As performance and scale of computer clusters and datacenters increase, the energy consumption of such facilities reached alarming rate and it is going to continue to rise in the future. While energy consumption increases, the utilization level of such facilities stays not so high level, or even lower than 50% in most cases [barroso2007case]. Moreover, it is very well known phenomenon that overall utilization level of most clusters has certain patterns over some period of time. For example, there are the peak and the lowest utilization periods. Since the utilization level of such facilities has a certain pattern and resides in a very low level during most of the day, turning-down or making unused portions of the system into low-power mode is one of the most straight-forward and effective means. In 2007, Barroso et al [6] introduced the term *Energy-proportional computing*, which delivers the idea of using only necessary amount of machines for a given workload at any time. Therefore, controlling machine power states actively as well as keeping the quality of service at or above the *service-level agreement* has become a new and challenging feature of any datacenters and computer clusters.

1.1.2 Popularity of the cloud computing

Recent years, cloud computing is attracting the attention of IT world with its high potential to change the way today's enterprises work. By using the cloud resources instead of possessing physical machines, cloud technology frees enterprises from the burden of taking care of physical machines locally.

Moreover, one of the fundamental features of cloud technology is the *Pay-as-you-go* payment policy. The user can use resources dynamically as her need increases or decreases. Since the user pays for the amount of resource she used, it gives the cloud service users an opportunity to save costs. Therefore, requesting the exact necessary amount of cloud resources she needs is one of her best interests. There are various resource scale-out and scale-down mechanism being offered (e.g., [29], [28]) considering factors such as schedule, rule, performance requirements, user's budget concerns and workload prediction. When cloud service user's resource scales wisely, it is not only economically beneficial to them, but also cloud service providers can reduce their power consumption by turning-off idle resources. For these reasons, there are a number of tools already developed or in the process of development, e.g., Amazon's *Heat* auto scale[14], OpenStack's *Elastack* [7], as a part of their cloud management services. While scaling has become crucial part of both cloud service providers and users, to our best knowledge, there still is no generic cloud resource managing framework which users can apply for various kind of environments from different cloud service providers. Moreover, many enterprises use hybrid environments by utilizing both local physical resources and cloud instances. Since each cloud providers offer different scaling options, for those who operate on hybrid environment cannot have a single scaling module to apply on all resources. This lack of generic resource controlling platform is another motivation of our work.

1.1.3 Complex dependencies in resources

So far, we explained how important it is to control and provision the resources dynamically both on physical and virtual environment. However, it is common that there exist complex dependencies among services, servers and other kind of resources such as switches, storage devices. When there are dependencies among resources, it is hard to distinguish the resources we can turn-off from the ones we cannot. For example, on a HPC cluster where each slave servers are connected to a master server. Even the master is not necessary or being almost idle, we cannot turn it off since other worker servers are still running and turning the master would cause serious malfunction in the entire system. It might sound trivial, but when the number of resources, network configurations and dependencies increase, the complexity becomes huge. When there comes other kind of dependencies such as logical *OR* and *AND* operations, the situation becomes even more complex and determining the right nodes to turn-off/on becomes tedious. We discuss about dependencies in detail in Section 3.2.

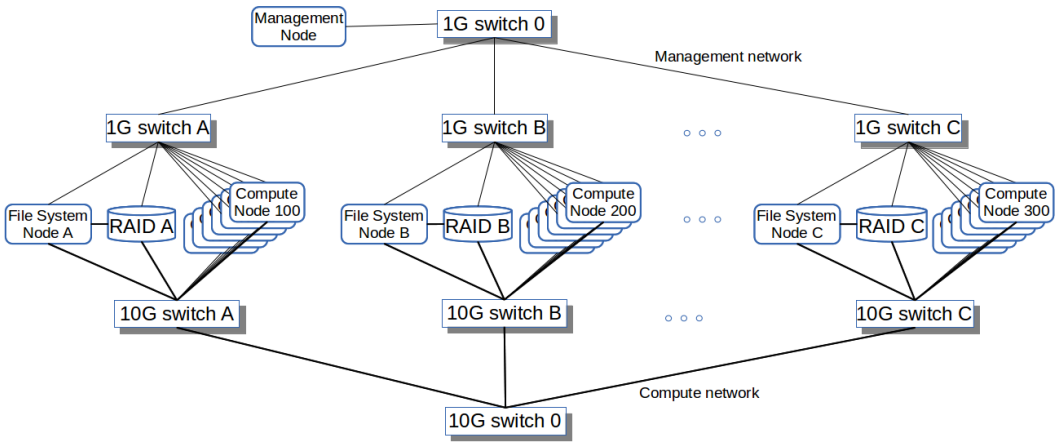

Figure 1.1. A typical cluster structure

In Figure 1.1, it shows a typical HPC cluster which consists of a *management node*, *compute nodes*, their *master nodes*, *Storage device*, and corresponding network devices , e.g., *switches*. To be running seamlessly, a compute node needs its corresponding master node, storage, and some switches that connects it with necessary nodes. We define this textitRUN-dependency between compute and master nodes. To turn-on, say, compute node, we might need these resources as well as the management nodes to be ON. We define this relationship between two nodes a *ON-dependency*. Same goes to *OFF-dependency*. There exists dependency not only between servers, but also between other resources such as services, storages, and switches. It makes dependency even more complicated to figure out which resource we can finally turn-on/off. We discuss about finding the best power state of system in Section 3.3.1.

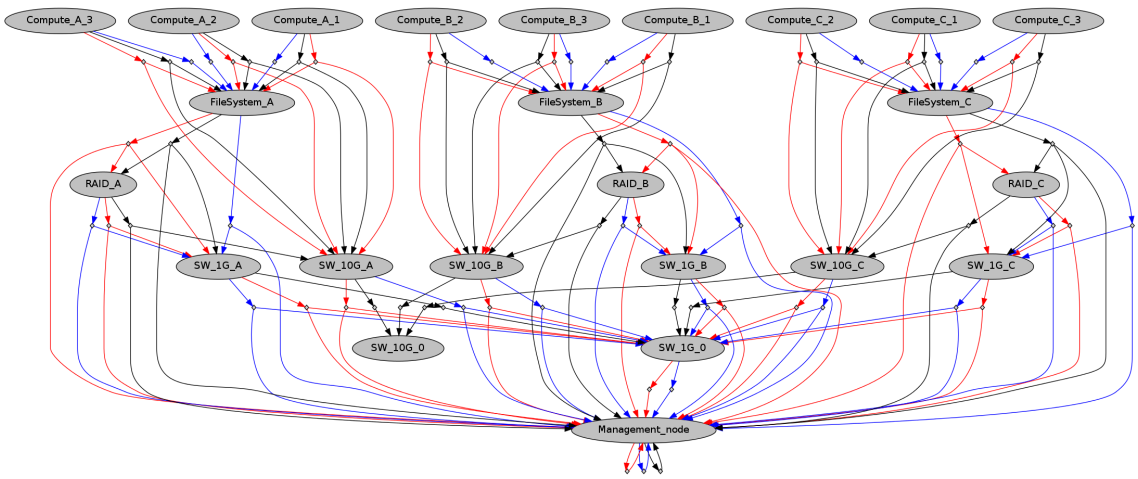

Figure 1.2. Simple (ON, OFF and, RUN) dependency graph of the cluster shown in Figure 1.1

When we take RUN, ON, and OFF dependencies of this simple cluster in Figure 1.1,

the dependency graph becomes as shown in Figure 1.2. While, determining the resources we can turn-on/off at give situation is considered complicated, to reach the desired power state of entire system gracefully is another non-trivial problem. In section 3.3, we introduce our proposal algorithm to solve this problem.

1.2 Contribution

As mentioned in the beginning of this section, with the exponentially growing requirement of dynamic resource provisioning, controlling the different kinds of resources from single interface while keeping the system reliability at the same time has become inevitable necessity. The main contribution of this paper is, first, we proposed a *Power State Transition Algorithm* for moving the system into low power/cost state gracefully by determining the best energy (cost) system state and reaching that state gracefully. Second, by exploiting our proposal algorithm, we implemented a generic power controller tool, *ClusterNap* [13], which can control in various kinds of systems and resources such as physical resources (servers, their switches, RAIDs and, CPU cores i.e,) virtual resources (e.g., Virtual machines, cloud instances). The demonstration of these usages are introduced in Chapter 5.

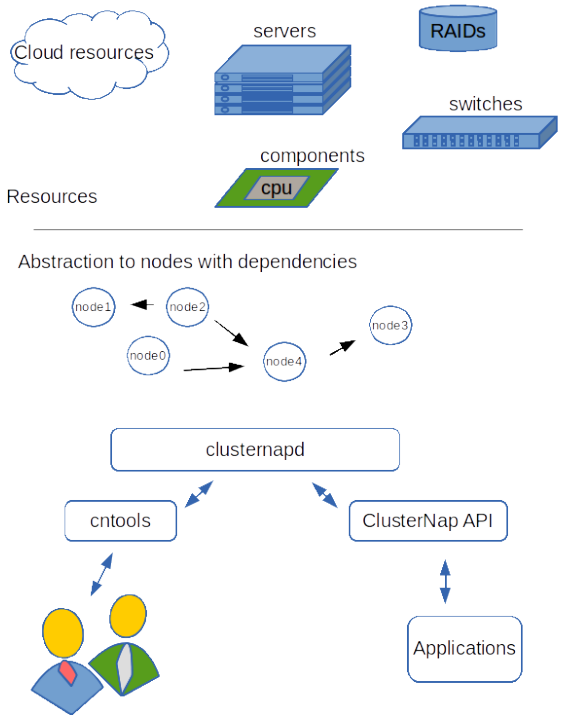

Figure 1.3. ClusterNap takes various kinds of resources as nodes. Users do not need to worry about the burden of taking care of differences among them.

1.3 Organization of the thesis

The rest of this paper is organized in the following order.

In Chapter 2, the related works are introduced. First, works conducted on energy consumption issue are mentioned. Next, works related with cloud resource provisinioning is introduced. At last, other resource management software are described.

In Chapter 3, our proposal *Power State Transition* algorithm is explained in detail.

In Chapter 4, by exploiting our proposal algorithm, implementation of resource power controller tool *ClusterNap* is described. ClusterNap currently consists of daemon called *clusternapd*, user interface tool *cntools* and its API. The detail of implementation is introduced in this chapter.

In Chapter 5, we conducted several experiments to demonstrate ClusterNap's usability in different kind of environments and resources. We demonstrated that ClusterNap can be used to controlling various resources including physical servers, RAIDs, switches, and even virtual instances and services. Moreover, it showed that ClusterNap can solve some complicated problems that can be reduced to *minimum-set-covering problem.*

In Chapter 6, we concluded our works conducted so far and suggest possible future works.

Chapter 2

Related Work

ClusterNap is designed to work either in physical clusters or cloud environments. There are various approaches to save energy/cost in these environments from the perspective of both users and resource providers. In physical clusters, for instance, the efficient level of energy consumption is saved by turning-off idle nodes heavily depends on the types of services (e.g., HPC platform, cloud resource, and web hosting) they provide. If the cluster runs as HPC platform, the frequency and the type of the jobs, the number of the users or even priority of each job should be considered carefully. Moreover, the power sources (e.g., central power-supply or smart-grid), time of the day (some power suppliers sell electricity at different costs for different period of the day), type of the resource (some resources consume more electricity than the others) can play important roles. There are too many factors one should consider in order to achieve the best result on power consumption saving and it is almost impossible to propose a single solution that works for all kinds of environments. Various related approaches to solve this energy consumption problem from the different facets of physical clusters are introduced in the Section 2.1.

Meanwhile, the rise of the cloud computing also gives rise to the importance of dynamic management of IT resources as well. The *elastic* management of the cloud resources has notably attracted the attention of many researches. Supposedly, gaining general approach on the perfect elasticity on one's cloud resource is far from the reality since each service users running on cloud resource have different characteristics. Those different approaches are introduced in the Section 2.2

2.1 Energy efficiency in physical clusters

The kind of computations running on a cluster has tightly related to selecting the nodes to turn-off. Maheshwari et al. [27], for instance, introduced a *data placement and cluster reconfiguration algorithm* for clusters running *Hadoop Distributed Filesystem (HDFS) and MapReduce framework*. When a cluster is running distributed filesystems such as HDFS, the filesystem makes replicas for each data blocks and distributes the replicas across the nodes for the better fault-tolerance. Moreover, systems such as Hadoop, the central master

node keeps track of all the worker nodes and replicas of the data blocks. When we turn-off some nodes for the sake of energy-efficiency, originally the master node assumes those nodes are broken, and starts to make replica of data blocks again. It induces unnecessary overhead in the system. In their work, Maheshwari et al. introduced such reconfiguration algorithm that makes the master node work properly even when some nodes are disable due to energy-saving. Moreover they also proposed a data block placement algorithm when some part of the cluster is OFF, and conducted the simulation. We should be aware that the purpose of ClusterNap is not parallel with their work, rather ClusterNap can be effectively used in their work to realize their simulation in real life clusters.

As mentioned before, the heterogeneity of a cluster nodes should be in consideration when we try to achieve the energy-efficiency. F.Coutinho et al. [11] proposes Hgreen heuristic (Heavier Task on Maximum Green Resources) and workflow scheduling algorithm for globally distributed clusters. Since the final step to save power consumption is to switch off unused nodes, ClusterNap can be used for their work, especially because it does not rely on physical location and can control various sites simultaneously.

When a cluster is running VMs as its service, migrating scattered VMs at certain part of the physical servers (i.e., server consolidation), and turning-off the rest of the nodes shows a favorable result on energy consumption. L.Hu et al. [18] proposed such implementation of a novel approach.

In the HPC world, its common to use systems of multiple clusters such as *inTrigger* [3], or Grid5000 [9]. In such systems, where users run their work on bare metals, having a *green policy* is crucial among any other factors. A.Orgerie et al. [31] proposed such policy in the Grid5000 system. Moreover, they created an *Energy-Aware Reservation Infrastructure (EARI)* where users reserve the physical resources along with selecting preferred work starting time and expected running time. EARI then schedules their reservation and turns-off the idle nodes.

Not only the compute nodes, as explained so far, but also the data storage devices consume significant portion (25-35%) of cluster's energy on average [15]. By considering basic functions to enable flexible energy management and data replication strategy, J.Kim et al. [24] [23] proposed an approach *Fractional Replication for Energy Proportionality (FREP)* for large datacenters. In this situation, there are basic two challenges when trying to gain energy-efficiency by turning the idle disks into low-power mode. The first one is the *Energy and response time penalty*; the second one is the *expected length of inactivity periods*. Their approach guarantees fault tolerance and better performance in large systems.

2.2 Cost efficiency in cloud computing

There are two ways to look at the technique of turning idle resources into low power mode in order to save costs or energy consumption in a cloud environment. First, it is from the

cloud resource user's perspective. Since the main characteristic of cloud service providers is the Pay-as-you-go cost policy, using the exact necessary amount of cloud resource becomes important for the cloud users. The related works from the cloud users' point of views are introduced in the Section 2.2.1.

Second, from the perspective of cloud service providers, it is also crucial to keep the utilization level of their physical resources high by switching off some part of resources. Reaching high utilization level by this way, however, tends to have trade-offs with the quality of the services they are providing to their users. Therefore, careful consideration with other facets of the situation is required. Studies being proposed in this situation are introduced in the Section 2.2.2.

2.2.1 Perspective from the cloud service users

When one uses cloud resources as a computational or service providing platform, using only the necessary amount is economically important. S.Niu et al. [30] proposed a *Semi-Elastic Cluster*, an on-demand resource provisioning computing model, for the ones who use HPC clusters on the cloud. Their work is similar to A.Orgerie et al.'s [31] work, but intended to work in HPC clusters in cloud. S.Niu et al. introduced integrated batch scheduling and resource scaling strategies. Our work is supposedly work well with their model since ClusterNap has shown that it can effectively control cloud instances.

Since on-demand resource provisioning is indispensable part of cloud services, the cloud service providers offer their own auto-scaling *elastic* solutions to their customers. *Amazon*, for instance, offers its own *auto scaling service* [14] to the customers of their cloud services. Their auto-scaling tool basically monitors system's load, and when certain conditions are satisfied, it turns-on/off *application servers*. Rackspace [21] and other cloud providers also offers their own elastic tools to their users. While they can be very effective measure for certain enterprises, these auto-scale tools are, however, suitable in very limited situation such as application servers. In addition, when a user is using hybrid environments such as using her own physical resources along with resources from different cloud service providers, there is no central controlling system. In this situation ClusterNap-like controllers can control all kind of resources from a single interface. We have to admit, however, that currently ClusterNap lacks such easy policy making features yet.

2.2.2 Perspective from the cloud service providers

While utilizing dynamic resource provision is economically beneficial to cloud users, cloud service providers also can reduce power consumption by dynamically controlling their physical servers' power states. Since cloud providers' main job is to provide virtual resources, which reside on top of their physical resource, for the users, migrating scattered virtual resources to certain part of the physical nodes and turning off the rest of the nodes can reduce the overall power consumption (server consolidation). While it might reduce power

consumption, there are other trade-offs (e.g., service degradation, migration overhead) because usually cloud infrastructure consists of several complex subsystems. Therefore, very careful consideration is needed and this topic has been in the focus of many researchers.

L.Beernaert et al. [7] proposed an automated monitoring and adaptive system, Elastack, which can work on existing IaaS providing software such as OpenStack [5], Eucalyptus [2], and OpenNebula [4]. Elastack enables the elastic feature on the clouds built on top of these infrastructure software [5] [2] [4]. C.Yang et al. [37] also proposes Green Power Management for supporting green power management in Cloud infrastructures, implementing VMs onto Opennebula, and integrating Dynamic Resource Allocation.

A.Beloglazov et al. [8], R.Buyya et al. [10] present current vision and challenges of the energy efficient management of the cloud computing environments. While considering the Quality of Service (QoS), they proposes energy-aware resource allocation and resource scheduling algorithms.

In cloud infrastructures where jobs are VMs, VM scheduling plays an important role. When it comes to energy-awareness, VM scheduling becomes even more important. K.Kim et al. [25] investigates power-aware VM provisioning for real time services. A.Younge et al. [38] also proposes similar approach.

S.He et al. [16] proposes lightweight resource management model Elastic Application Container(EAC) whereas traditionally it is VM. In addition to proposing new virtual resource unit, they also proposes energy-efficient resource provisioning algorithm.

We have mentioned works in above subsections because dynamic resource provisioning and management has already become a crucial part of today's IT systems. In the very bottom of these dynamic behaviors of these systems, there must be certain interface which controls the state of each unit of resource (e.g., VMs, instances, servers, storages, and switches). Unfortunately, there is no single generic interface or platform to control these various types of units. When there are complex dependencies among these units, it becomes even more complicated. ClusterNap's main advantage is that it controls such various types of resources as long as the user has defined them.

2.3 Similar tools

There are several power controller tools which have similar features as ClusterNap. S.Kiertscher et al. [22], for instance, proposes an energy saving daemon called *Cherub*. Cherub works in environments which have central resource management structure. While ClusterNap controls various kind of nodes such as server and switches Cherub controls only compute servers. Cherub considers a number of server states, i.e., *Unknown, Busy, Online, Offline, Down*. On the other hand ClusterNap has three kinds of states; *On, Off, Unknown*. Cherub takes four different actions according to which state it want a node to transit; *try_boot, try_register, try_sign_off* and *try_shutdown*. ClusterNap, instead, takes only two kind of actions by running defined scripts; *try_on* and *try_off*. Similar to ClusterNap,

Cherub works with the Torque batch scheduler and it has simple API. Another similar approach with Cherub is *Roll* proposed by M.Dolz et al.. It, however, is dedicated for only *Sun Grid Engine* and related services. K.Kurowski et al. [26] proposes a distributed solution called *SMOA* for monitoring and managing computer systems. SMOA is similar to the tools introduced in this section, however, it does not only control power states, it also monitors power consumption of each nodes. In addition, to transfer information it uses eXtensible Meesage and Presence Protocol (XMPP) so that it can cope with heavily firewalled environments.

Adaptive Computing [1] has recently developed their auto power management module for their workload manager *Moab* [35]. Moab's power management has two node power states; *On* and *Off*. The basic structure is also similar to ClusterNap; the user defines state changing scripts for each node and the power management unit executes them when necessary. Differing from ClusterNap, Moab has its own node state checking daemons running in the worker nodes. Each daemon then sends its host's state to the master node. It also has its own green policy maker, while ClusterNap currently lacks such policy making functionality. The power management module is only available for Adaptive Computing's priced products and it is not available for free. There are also other vendor specific similar tools such as HP's *iLO* [17], DELL's *DRAC* [12], or IBM's *EndPoint* [19]. These tools are highly integrated with corresponding resource monitoring and management systems. Moreover, they all are dedicated to only their specific environments and not suitable for other systems.

2.4 Dependency and resource allocation problems

The most unique feature that no other similar tools have is ClusterNap's resource dependency-awareness. Once a user defines the dependencies for each node, ClusterNap takes care of the rest and executes necessary state changing scripts according to the dependency so that the system avoids possible system malfunction. When there are complex dependencies, choosing the best (e.g., combination of nodes which have the most energy or cost efficient, as well as fulfills the user's requirement) resources is complicated problem (see the Chapter 3). One example pf the very similar problems is the Linux's *Minimum install problem*. Since, in Linux OS, packages have their dependencies, there are multiple ways to install a given package(s). Finding a way to install minimum size (or number) of packages can also be reduced to *Pseudo-Boolean problem*. One of the available solutions is proposed by Tucker et al. [36]. They proposed such package install manager called *Opium* which chooses the best combination of packages when a user requested certain package(s).

Chapter 3

Power State Transition Algorithm

3.1 The main problem

When the system's utilization level is low, turning node's power status into low power consuming state in cluster system is much more complicated than in a system which consi sts of a single, standalone node.

First, the operator must guarantee the Quality of Service (QoS) she agreed to provide to users must be in the range of certain level. In other words, services the (cluster) system provides have to continue and be in certain quality level regardless of the power states of the nodes in that system. In addition, the system as a whole should be stable enough regardless of the power states of the nodes. Not only should enough nodes be alive at any given time, but also the *right nodes* must be alive. System's power controller must correctly determine those nodes to keep the system stability and it is not a trivial task for most clusters, since there are most probably very complex dependencies among the resources (both physical and virtual) and services. Finding the *minimum power state* of a cluster, when certain nodes are requested, can be reduced to *Pseudo-Boolean constraints* (Figure 3.1.)

$$min : k_a \times node_a + k_b \times node_b + ... + k_z \times node_z$$
$$s.t : Dependency(D) \wedge requested\_nodeA \wedge requested\_nodeB...$$

Figure 3.1. Minimum Power State finding problem can be reduced to Pseudo-Boolean problem. It is considered NP-Hard

When nodes' dependency $Dependency(D)$ and requested nodes (i.e., $requested\_nodeA$, $requested\_nodeB$) are given, we can generate pseudo-boolean constraints shown in Figure 3.1, where $k_i(i = a, b, c...)$ is, for instance, the power consumption (or *cost* in the case of cloud resources) coefficient of each nodes. Nodes can take a value of $1(ON)$ or $0(OFF)$. Finding the satisfying assignment, which results in minimum power consumption, is thus, in above case, considered NP-Hard problem. We use simple greedy algorithm to find the approximate optimal solution and it is described in Section 3.3.1.

Second, even if we know the *minimum power state*, finding the correct path to reach that state is also improtant. To reach the minimum power state gracefully, we need to turn-on and turn-off the right nodes in right orders. Our proposal *Power State Transition algorithm* solves this problem and is described in Section 3.3.2.

3.2 Node dependencies

As mentioned in Section 1.1, a node is a single resource entity such as physical resource (i.e., server, management node, storage, network switch), virtual resource (i.e., VM, cloud instance) or service (i.e., certain service or a state of being able to connect to a server or access data). A node can have 3 different power states; **Running** (or **ON**), **Shut-off** (or **OFF**) and **Unknown**. There are 3 different dependencies we consider in our algorithm; **RUN-dependency**, **ON-dependency**, and **OFF-dependency**. All dependencies are expressed in *Conjunctive Normal Form* of Boolean Logic.

RUN-dependency

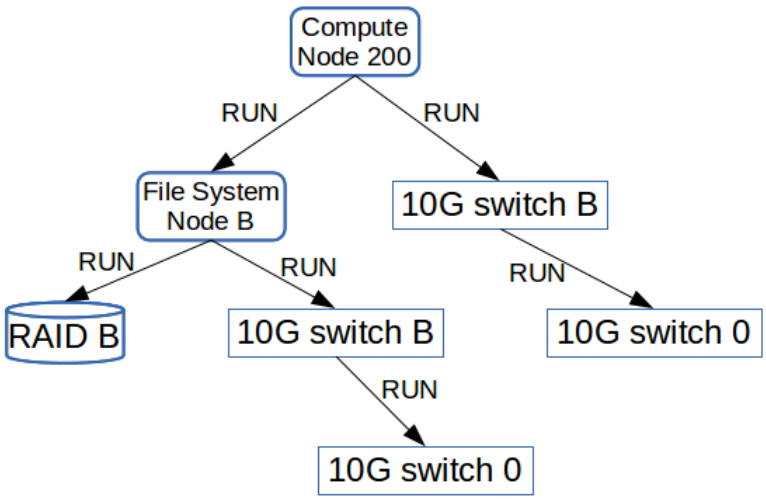

Figure 3.2. Simple dependency

We define RUN-dependency as following:

Definition 1 (RUN-dependency)**.** *If NodeB has to be ON to keep NodeA, we say there is RUN-dependency between NodeA and NodeB. Alternatively, we say NodeA is RUN-dependent on NodeB. RUN-dependency is denoted as* $NodeA \xrightarrow{RUN} NodeB$.

For example, on a cluster shown in Figure 1.1, *Compute Node 200* might be RUN-dependent on *Filesystem Node B* and *10G switch B* (Figure 3.2). Furthermore, *Filesystem Node B* can be RUN-dependent on *RAID B* and *10G switch B* since their relationship fullfils the definition of RUN-dependency.

ON-dependency

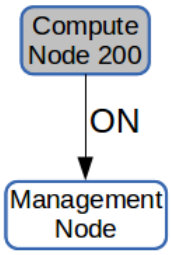

Figure 3.3. ON-dependency

Likewise, we define *ON-dependency* in below.

Definition 2 (ON-dependency)**.** *If $NodeB$ has to be ON to make current OFF node $NodeA$ ON, we say there is ON-dependency between $NodeA$ and $NodeB$. Alternatively, we say $NodeA$ is ON-dependent on $NodeB$. ON-dependency is denoted as $NodeA \xrightarrow{ON} NodeB$.*

For example, to turn-on $ComputeNode200$ in Figure 1.1, we might need $ManagementNode$ to be ON, beforehand, and that $ManagementNode$ might not need to be ON once our requested server becomes ON. Therefore, we represent this situation as ON-dependency (Figure 3.3)

OFF-dependencies

Definition 3 (OFF-dependency)**.** *If $NodeB$ has to be ON to make current ON node $NodeA$ OFF, we say there is OFF-dependency between $NodeA$ and $NodeB$. Alternatively, we say $NodeA$ is OFF-dependent on $NodeB$. OFF-dependency is denoted as $NodeA \xrightarrow{OFF} NodeB$.*

A similar example as ON-dependency can be applied to OFF-dependency.

AND relation in a dependency

Definition 4 (AND relation in a dependency)**.** *When node $NodeA$ is dependent on more than one node $NodeB, NodeC$... at the same time, we say there are AND relation among $NodeA$'s dependency node. We denote and relati'on $NodeA \rightarrow$ ($NodeB$ AND $NodeC$ AND ...)*

For example, in Figure 3.2, $ComputeNode200$ is RUN-dependent on both $FilesystemNode$ and $10GswitchB$ at the same time. Therefore we can write this dependency as:

$$ComputeNode200 \xrightarrow{RUN} (FilesystemNode \textbf{ AND } 10GswitchB).$$

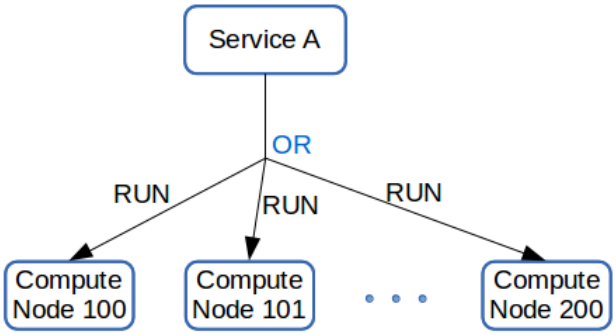

Figure 3.4. OR relation in dependency

OR relation in a dependency

Definition 5 (OR relation in a dependency)**.** *When node $NodeA$ is dependent on either of several groups of nodes $NodeGroup1, NodeGroup2...$ at a time, we say there is OR relation among $NodeA$'s dependency groups of nodes. We denote the relation as $NodeA \rightarrow$ ($NodeGroup1$ OR $NodeGroup2$ OR ...).* ***A node group*** *can consist of one or more nodes with AND relations, i.e., NodeGroup1 = { NodeB AND NodeC ...}.*

As an example of OR relation, consider a case when a certain service can be run when only one of $ComputeNode100$ **OR** $ComputeNode200$ is ON, we can represent this relation by OR as define in Definition 5 (see Figure 3.4)

In this thesis, we call a node group a **clause**. Therefore, in further parts of this paper, a clause refers to a group of nodes with AND relations among each other. Moreover, with the three kinds of dependencies and two relations introduced in this section, we can write the dependency nodes in *Conjunctive Normal Form (CNF).*

3.3 Power State Transition Algorithm

We have discussed the main problems of the power transition method in the Section 3.1. In addition, the main terminologies that are used in the State Transition Algorithm are also defined and explained.

The main flow of State Transition Algorithm (STA) is shown in Figure 3.5. First, STA gets the information of the *current power state* of the system **State_current**, requested nodes (nodes those are requested by the user) **R**, and the dependency of the system **D**.

With these three values – **State_current**, **R**, and **D** – we first need to determine the minimum power state (power state on which the system consumes the least energy and still be able to continue to provide its service gracefully) **State_min** to which our system need to reach. How we determine the **State_min** is explained in Section 3.3.1.

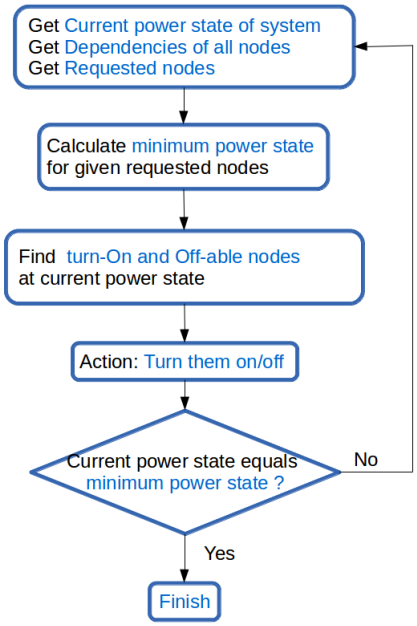

Figure 3.5. Main algorithm

Once the State_min is determined, we need to take corresponding actions to approach that state. Therefore, based on D, R, and State_current, our algorithm determines the nodes which we can turn-off, **Nodes_to_off**, and the nodes which we can turn-on, **Nodes_to_on**, to approach the State_min. In Section 3.3.2, we explain how Nodes_to_on and Nodes_to_off are selected.

After determining Nodes_to_off and Nodes_to_on, the power controller tries to turn off Nodes_to_off and turn-on Nodes_to_on.

When the power controller changed power states of some nodes, now the State_current is changed. If the new State_current equals State_min, the algorithm finishes. Else, the algorithm starts from the beginning until the system reaches the State_min.

3.3.1 Minimum power state

Minimum power state is a power state of a cluster that consumes the least power possible while still keeping the user's requested nodes running seamlessly. When the dependency **D_run** and requested nodes **R** are given, we can construct a Pseudo-Boolean constraint as shown in Figure 3.1. In this case, solution of this problem is a groupd of nodes which *satisfies* the boolean function on the second line of Figure 3.1 while keeping the value of equation in the first line *minimum*. As mentioned in Section 3.1, finding these group of nodes is considered a NP-hard problem. Fortunately, there are several approximate Pseudo-Boolean constraint solvers have been proposed. A similar problem is Linux's *Minimum Install problem*. In the Linux OS, packages have their dependencies (i.e., to install one package, other packages might be needed to be installed beforehand), and

```
0  def minimum_power_state(D, R):
1
2     State_min = R
3     while R != []:
4        N = choose_node_from(R)
5
6        # D[N] is in Conjunctive Normal Form
7        for clause in D[N]:
8           for node in clause:
9              if node in State_min:
10                # Remove node from D[N]'s all clauses
11                remove_from(D[N], node)
12
13       State_min += choose_smallest_clause(D[N])
14       R.remove(N)
15
16    return State_min
```

Figure 3.6. Minimum-power state finding algorithm

there are various ways to install a given package. Finding the way to install minimum size (or number) of packages can also be reduced to Pseudo-Boolean problem. One of the available solutions is proposed by Tucker et al [36]. In our case, we are using a simple greedy way to determine the group of nodes for minimum power state. The algorithm is shown in Figure 3.6.

This algorithm is somewhat similar to *Greedy set-covering algorithm* (Appendix B) introduced in Leiserson et al's book [32]. In our algorithm, we choose the least number of dependency nodes of a requested node and put them into State_min node list. Of course, the dependency nodes those are already in State_min is excluded. This algorithm tries to get the least number of nodes that satisfies the second line of the boolean equation in Figure 3.1 by greedy manner.

3.3.2 ON/OFF-able nodes

By applying the *Minimum power state algorithm* as well as given the current power state and requested nodes, we can calculate which power state we should try to reach. However, to reach the destination state State_min, we cannot just turn-off ON nodes that are not in State_min and, turn-on OFF nodes that are included in State_min since there are also ON, OFF and RUN dependencies for each of those nodes. Therefore, we need to know exactly which nodes we should turn-off and which one we should turn-on at the moment without causing any inconvenience to the system.

In Figure 3.7, the procedure of determining turn-on and off-able nodes (Node_to_on and Nodes_to_off) is described.

```
def on_off_nodes(D_run, D_on, D_off,
                 State_min, State_current):

    Nodes_to_on  = Nodes_off & State_min

    Nodes_to_off = Nodes_on - Nodes_to_on

    # '***' is Cartesian product operation
    Nodes_to_on += minimum_power_state(D_run *** D_on,
                                       Nodes_to_on)
    Nodes_to_on += minimum_power_state(D_run *** D_off,
                                       Nodes_to_off)
    Nodes_to_on  = Nodes_to_on - Nodes_to_off

    # Exclude non-turn-on-able nodes
    for node in Nodes_to_on:
        # If there is at least one clause of "node"'s
        # RUN-ON dependency has all members ON
        if not any(all_members_on(clause) \
                        for clause in D_run_on[node]):
            Nodes_to_on.remove(node)

    # Exclude non-turn-off-able nodes
    for node in Nodes_to_off:
        # If there is at least one clause of "node"'s
        # OFF dependency has all members ON
        if not any(all_members_on(clause) \
                        for clause in D_off[node]):
            Nodes_to_off.remove(node)

    # We cannot turn-off any node on which
    # another ON node is dependent on
    for node in Nodes_to_off:
        if another_on_node_dependent(node):
            Nodes_to_off.remove(node)

    # Call the same function until the Nodes_to_on is
    # all included in State_min
    if not State_min <= Nodes_to_on:
        return on_off_nodes(D_run, D_on, D_off,
                            State_min | Nodes_to_on,
                            State_current)

    return Nodes_to_on, Nodes_to_off
```

Figure 3.7. Pseudo-code of a function to determine On/Off-able nodes

First, in line 2 -6 in Figure 3.7, we are determining what other nodes are required to be ON to make OFF nodes ON that are in State_min by using Minimum power state algorithm. Furthermore, nodes that are required to be ON to make unnecessary ON nodes OFF are also determined here. Each of these two kind of nodes are now added to the list of Nodes_to_on and Nodes_to_off respectively. In line 8, *** is an operation where *Cartesian product* of each node's RUN-dependency and ON-dependency nodes are taken (each dependency is in CNF) and returned in CNF. Same logic applies to the *** operation in line 10.

We cannot make all nodes in Nodes_to_on ON at the same time due to dependencies. For example, in Figure 1.1, let us assume all of $FilesystemNodeA$, $RAIDA$, and $ComputeNode100$ are OFF. When the node $ComputeNode100$ is requested, all three nodes are put in the State_min list according to the RUN-dependencies among them. However, as mentioned above, we cannot just turn-on all three of these nodes at the same time. According to RUN-dependency, we should, first, turn-on $RAIDA$. Once we know that $RAIDA$ has become ON, we now can turn-on $FilesystemA$. Likewise, after confirming $FilesystemA$ has become ON, finally we now can turn-on $ComputeNode100$. Therefore, only Turn-on-able node is $RAIDA$ and other two nodes should not be included in Nodes_to_on list, when all three of them are OFF (line 8-12 in Figure 3.7).

Similar logic applies to turn-off-able nodes for Nodes_to_off. For example, let us assume that all three of $FilesystemNodeA$, $RAIDA$, and $ComputeNode100$ are ON and we do not need them anymore, so we should turn them off. However, we cannot just turn-off them all together instantly. We should turn-off these nodes in the *right* order; i.e., first $ComputeNode100$, then $FilesystemNodeA$, and finally $RAIDA$ (line 15-19 in Figure 3.7.) Furthermore, we should check if another ON node is dependent on any of these three nodes, in that case, we must not turn-off that node (line 32-34 in Figure 3.7).

3.3.3 Action

Once we determine Nodes_to_on and Nodes_to_off, now power controller does its job – to change their power states. Since each kind of node can be turned-on/off by a different ways, this part should be configured by the system's owner. We propose our solution to this state change way in ClusterNap in the next Chapter 4.

By repeating the steps explained through Section 3.3.1 - 3.3.2, cluster's power state State_current gradually approaches to State_min. In reality, some nodes might have *Unknown* state especially during when the power state is being changed from ON to OFF or vice versa. Our algorithm also considers this case and takes *actions* based on the states of the nodes which are only ON or OFF.

Chapter 4

ClusterNap

ClusterNap is a simple generic power control tool for clusters and cloud infrastrucutures. ClusterNap basically implements our Power State Transition algorithm proposed in Chapter 3 and executes user defined ON/OFF commands to change nodes' power states. Therefore, only by defining 3 kind of dependencies and ON/OFF commands in the corresponding configuration file, ClusterNap determines which nodes should be ON and which ones can be OFF. Furthermore, it executes those commands in the right order while keeping the system stable.

Since, to ClusterNap, a node means any type of resource the user has defined such as a physical server, a network device, or a cloud instance, ClusterNap actually can control these nodes from various kinds of environments. The implementation consists of three main parts; daemon *clusternapd*, user interface *cntools*, and *ClusterNap API*. In the following sections 4.1 - 4.3, we discuss about these three parts in detail.

4.1 ClusterNap daemon: clusternapd

As we have seen in Chapter 3, our State Transition Algorithm takes State_current, requested nodes R, and three kinds of dependencies (ON, OFF, and RUN) as arguments and returns the nodes that we can turn-on/off (Nodes_to_on and Nodes_to_off) at the current state. The daemon *clusternapd* executes our proposal algorithm and gets Nodes_to_on and Nodes_to_off. Once it gets these on and off-able nodes at the current state, it runs corresponding on and off scripts which user defined in the configuration file. The main flow of clusternapd is depicted in Figure 4.1.

clusternapd runs every t seconds (by default t is 5 seconds, and the user can change it). As long as the arguments (e.g., current state, requested nodes, dependencies) do not change, clusternapd executes the same on/off scripts. When the scripts executed by clusternapd actually changes the power state of nodes, the current power state of the system, State_current, changes. After certain time t (by default 5 seconds) clusternapd runs the State Transition Algorithm again and gets new Nodes_to_on and Nodes_to_off because supposedly State_current has been changed due to result of the execution of

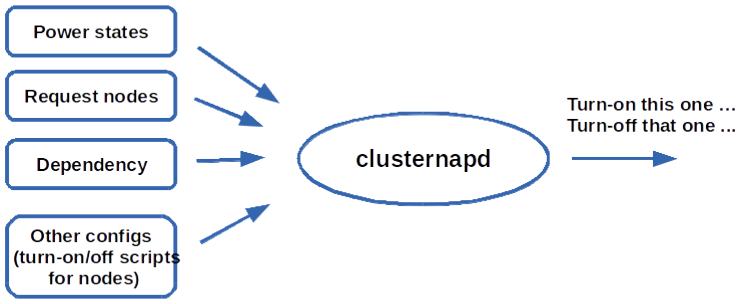

Figure 4.1. ClusterNap daemon, clusternapd's structure

the state changing scripts clusternapd has run previously. Again, clusternapd executes corresponding state changing on/off scripts every t seconds until the system reaches the minimum power (cost) consuming state, State_min. Once the State_current equals to State_min, Nodes_to_on and Nodes_to_off becomes empty and clusternapd executes no script while still running the State Transition Algorithm every t seconds.

4.1.1 System configuration and syntax

As arguments, clusternapd takes system's current power state, requested nodes, and nodes' dependency and their state changing script (Figure 4.1). To define the all nodes in the system, we use following syntax as shown in Figure 4.2.

If noticed, it is very similar to the configuration syntax of nodes in Nagios [20] monitoring system. In the line 1 in Figure 4.2, the *node name* is written. Line 2-4, three kind of *dependencies* are written. As mentioned in the previous chapter, dependencies should be represented as Conjunctive Normal Form, i.e., (NodeA AND NodeB) OR (NodeC AND ...). In other words, the dependencies are the clauses of nodes connected by AND relation connected by OR relation. In ClusterNap's config file, AND is denoted by ',' (comma) and OR is denoted by '|' (vertical bar). For example, in Figure 4.2, NodeA is RUN-dependent on **(NodeA AND NodeC) OR (nodeD)**. Particularly, the sufficient condition for node NodeA running properly is when both NodeB and NodeC are ON or NodeD alone is ON. The same logic applies to other two dependencies, ON and OFF-dependency.

In the line 5 and 6 in Figure 4.2, state changing (on to off and off to on) *commands* of NodeA is written. These state changing commands consist of four parts. First, the *node name* on which the command is executed is written. In our case, NodeD and NodeE are written respectively. Next, after comma, the *user name* by which the command is executed is written. In our case it is the user *root*. Next, the state changing *command* name is written. In the Figure 4.2, on-command, to make NodeA ON when it is OFF, is `ipmi_on`. In addition, the name of the command that makes ON NodeA to

```
0  define node{
1          name:                NodeA
2          run_dependencies:    NodeB, NodeC | NodeD
3          on_dependencies:     NodeD
4          off_dependencies:    NodeE
5          on_command:          NodeD, root, impi_on!ipmi_ip
6          off_command:         NodeE, root, ssh_off!ip_of_NodeA
7  }
8
9  define_command {
10         name:              ssh_off
11         command_line:      ssh $ARG1$ shutdown -h now
12 }
13
14 define command {
15         name:              ipmi
16         command_line:      ipmitool -H $ARG1$ -U root power on
17 }
```

Figure 4.2. Example of configuring nodes and state changing scripts/commands

OFF is `ssh_off`. These commands are defined by the user as well. After the '!' (exclamation mark), the argument(s) to the corresponding command is written. If more than one argument must be provided, they should be split by exclamation mark '!', e.g., `command_name!argument1!argument2!...!argumentN`.

Commands are also defined by the users. Since each node can have different properties, to make it ON or OFF, it might require quite different command from the other nodes. To make a server ON, ipmi's commands might suffice for some environment. For network switches or storage devices , however, other kind of scripts are required to be executed to change their power state. In the line 9-11 and 14-16 in Figure 4.2, commands to make node NodeA on/off are defined. The command to make node NodeA off has ClusterNap name `ssh_off`. To execute this command in the system, we run

```
ssh ip_of_NodeA shutdown -h now
```

on the node `NodeE` as a user `root` (line 6 and 11 in Figure 4.2). By defining a state changing commands this way, first, a user can use a single command for multiple nodes only by changing their arguments. Second, it gives the user freedom to choose state changing scripts. Instead of `ssh` or `ipmitools`, it can be any script or executable that the user chooses. In case of cloud instances, it can be a script that controls cloud instance's state. For network switches, it can be certain *snmp* command. For services defined as node, it can be a command to start/stop that service.

Users can write configuration of each node and command either in separate files, or in the same file as long as the config file(s) has the extension '*.conf*'. For example, a

```
define node{
  name:                 worker[000-099, 200-210]
  run_dependencies:     ...
  on_dependencies:      ...
  off_dependencies:     ...
  on_command:           head_node,root, \
                        ipmi_on!AA.BB.CC.[200-299, 400-410]
  off_command:          head_node,root, \
                        ssh_off!DD.EE.FF.[123-222, 005-015]
}
```

Figure 4.3. Defining multiple nodes in a single definition block

user can define all her servers in a config file named *servers.conf* and commands in file *commands.conf* and so forth according to the type of her resources.

Another convenient feature of its syntax is, users can define the same type of multiple resources in a single definition block as shown in Figure 4.3. In Figure 4.3, for example, a hundred nodes named worker000 - worker099 and ten nodes name worker200 - worker210 are defined. ClusterNap parses the items inside the brackets and generates each node names. Moreover, it parses the items inside other brackets as well and makes one-to-one correspondence to these generated items. In other words, ClusterNap anticipates $i - th$ element in each bracket corresponds to each other. For example, in the example shown in Figure 4.3, `worker014` corresponds to IP of `AA.BB.CC.214` and `DD.EE.FF.137`. Therefore, the number of items expressed inside all brackets have to equal to each other. Otherwise, we cannot make one-to-one correspondence.

4.1.2 Node state check: Nagios, Torque

Besides node dependencies and state changing scripts, clusternapd also takes current power state of system, `State_current`, as an argument. ClusterNap has three kinds of node states; on, off, and unknown. Currently, ClusterNap does not have its own node state checking module. We are, however, working on this module and expect to complete this module in the near future. Instead, for now, we uses other resource monitoring software as a node state checking module. In the system which uses *Torque* [33] resource manager, we implemented a feature that takes Torque's node states and updates ClusterNap's own node states. To enable this feature, the user only need to set environmental variable `CHECK_TORQUE` as *True* she starts clusternapd. Apparently, names of the nodes defined in Torque resource manager should match with the ones defined in ClusterNap.

In systems which use Nagios [20] resource monitoring system, we propose certain nagios node checking plug-ins. When a user uses these plug-ins to check their resources, instead of the Nagios' default host checking plugins, our plug-ins updates the resource states of Nagios, as well as those of ClusterNap. Since with Nagios, we can check the states of various kinds of resources, both physical and virtual, it is convenient way to update various resources of ClusterNap.

4.2 ClusterNap user interface: cntools

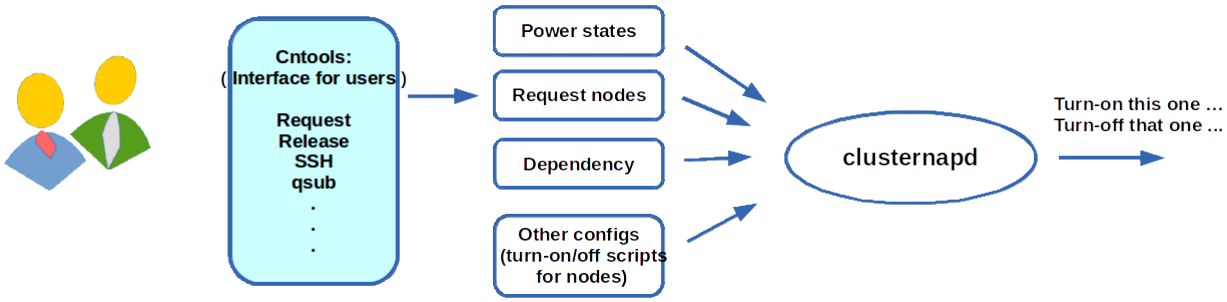

Figure 4.4. Structure of cntools

As shown in Figure 4.1, clusternapd takes current power state, dependency and state changing scripts, and **requested nodes** as arguments. In Section 4.1.1, we introduced how we configure nodes and state changing scripts. Once we configure them in the config file(s), clusternapd gets the necessary information from these config files. In Section 4.1.2, we introduced the ways we use to update system's nodes' states. The last argument is the *requested nodes*. Requested nodes are nodes requested by the users or the system. The main rationale behind ClusterNap, like other similar tools, is to use only necessary amount of resources. Therefore, when users want to use resource, they request the resources in necessity. According to these requested resources, clusternapd makes them available by turning-on right resources in right order. After the users conducted their works, they also might tell ClusterNap that they finished their works and release the resources. In order to make this process of requesting and releasing resources easy, ClusterNap has user interface tool **cntools**. The Figure 4.4 depicts the main structure of *cntools*. Once the user installed ClusterNap, she can use this interface by running the command

```
cntools <arguments>
```

in the command line. In the following sections, the details of cntools and its arguments are explained.

info

When a user runs command

```
cntools info
```

it displays the information of all nodes defined in the ClusterNap and respective power states. Moreover, it also displays which nodes are requested and which are not. If a node is requested, the user's name and the date of the request is also displayed. Figure 4.5. With the corresponding options, it also displays only `On` or `Off` or `Unknown` nodes; `Requested` or `Free` (not-requested) nodes; nodes requested by only given user.

```
 ---------------------------------------------------------------------------------
|   Node name    | Power state | Request state | Request user |   Request date    |
 ---------------------------------------------------------------------------------
|worker1         |Off          |Free           |N/A           |N/A                |
|worker3         |Off          |Free           |N/A           |N/A                |
|worker4         |Off          |Free           |N/A           |N/A                |
|master          |On           |Free           |N/A           |N/A                |
|worker2         |On           |Requested      |amgaa         |2014-01-23 12:34:56|
|localhost       |On           |Requested      |root          |2014-01-20 15:54:25|
 ---------------------------------------------------------------------------------
```

Figure 4.5. Sample output of the command `cntools info`

As seen in Figure 4.5, we can see the node `master` has power state `ON` even it is not requested by anyone. It is because node `node2` is dependent on `master` and ClusterNap keeps `master` ON. Once the user who requested `worker2` releases it and there are no other nodes requested, ClusterNap would turn-off `worker2`.After confirming that `worker2` has become OFF, ClusterNap then turns-off `master`.

release, request

By running the command

```
cntools request <nodename>
```

or

```
cntools release <nodename>
```

a user can request or release a node. When a user requests a node and if the requested node is not requested by anyone else (or `Free`), cntools confirms that request. Of course, if the node is requested by the user or someone else, cntools does not confirm the request and returns false. The same logic applies when a user releases a node. If the user tries to release a node someone else is requested, cntools does not confirm the request and returns false. cntools release a node only when the node is previously requested by that user herself.

When `requesting/releasing` multiple nodes, the user can execute a commands, for example, `cntools request/release nodename1 nodename2 nodename3`. Furthermore, if the user requests multiple nodes whose names share the same prefix with suffices in increasing order, she can use, e.g., following command.

```
cntools request worker[1-4]
```

In this particular example above, `cntools` will request the nodes `worker1`,`worker2`,`worker3`, and `worker`.

ssh, rsyn, scp, qsub

ssh, scp, rsync. `cntools` also provides other commands, i.e., `ssh`, `rsync`, `scp`, `qsub`. All these commands require certain resources (e.g., destination host or computation server).

In systems which use ClusterNap, resources can possibly be unavailable, say not requested, therefore OFF. Users, on the other hand, does not necessarily have to have all the power states of resources. For example, when a user wants to log-in to a host, she might not necessarily have to have the knowledge of the state of her destination host. Instead, if she uses a command

```
cntools ssh user-name@destination-host
```

cntools checks the host. If the host in question is available, cntools connects the user to that host. On the other hand, if the destination host is OFF, cntools this time requests that host. After requesting the destination host, cntools waits for some time t_{wait}. During this period, if the requested destination host become available, cntools connects the user to that host through ssh. ClusterNap does not simply turn on the requested node. If there is dependencies such that other nodes also should be ON, ClusterNap turns them on in right order. If the destination host does not become available in certain time (15 minutes by default) cntools returns false. In this way, without knowing the actual state of the requested resource, user can connect to the host.

The same logic applies to `rsync` and `scp` commands as well. Without knowing the corresponding resource's state, user can copy files with `rsync`, `scp`.

qsub. Many systems uses resource managers such as Torque [33] to utilize their resources efficiently. We mentioned in Section 4.1.2 that ClusterNap can work with Torque resource manager to update its resources' states. ClusterNap works with Torque's job submission command `qsub` as well. When users use `cntools qsub` with required resources specified, instead of `qsub` alone, ClusterNap gets required resources and requests them. If the required resources for the job submitted by `cntools qsub` are not available, eventually ClusterNap makes these resource available by executing necessary state changing scripts. Once the resources are available, resource manager's (in our case Torque) scheduler executes the submitted job.

Also, if the user sets environment variable `PBS_RELEASE` to `True` when starting the daemon clusternapd, ClusterNap `releases` the resources which are once required by Torque job, but not required anymore. In this way, the system can use only required amount of resources at given time.

Currently, cntools has these features mentioned above; info, ssh, scp, rsync, and qsub. User, however, can create their own **ClusterNap aware tools** by utilizing the ClusterNap's API which is introduced in the next section.

4.3 ClusterNap API for python

In systems which use ClusterNap, user might want to write their own *ClusterNap aware* programs like *cntools* introduced in the previous section. In that case, ClusterNap proposes its own *Application programming interface (API)* for python. By importing `cntools` in their python scrips, users can create clusternap object as follows:

Function name	Description
`get_dependency(nodename, dep_type)`	Returns the specified dependency of a node.
`get_info()`	Returns the information of all nodes defined in ClusterNap.
`get_node_info(nodename)`	Basically the same as get_info()[nodename] .
`get_power_state(nodename)`	Returns the power state of a node.
`get_request_state(nodename)`	Checks if a given node is requested or free.
`get_request_date(nodename)`	Returns the date the given node is requested, if requested.
`get_request_user(nodename)`	Returns the user who requested given node, if requested.
`release(nodename)`	Tries to release given node.
`request(nodename)`	Tries to request given node.
`set_power_state(nodename)`	Manually sets a node's power state. Useful when creating independent power checking module.
`set_dependency(nodename, dep_type, dependency)`	Sets new dependency for the given node and its specified dependency (ON, OFF, or RUN). Cannot set dependency if multiple node is defined in a single definition block.

Table 4.1. Functions of ClusterNap API

```
import cntools
clusternap = cntools.clusternap()
```

Now, for example, if a user wants to print the power state of a node, say, `'worker1'`, she can write inside her program

```
print clusternap.get_power_state('worker1')
```

ClusterNap's API has a function to return power state of a node `get_power_state()`. Entire list of the API's functions are shown in the Table 4.1.

Chapter 5

Experiments and Application Scenarios

ClusterNap is designed to be able to control various types of nodes and to consider their dependencies among each other. In addition its interface tool `cntools` and API enable it to be applied for various purposes. In this section we introduce some of its application examples and results.

5.1 `cntools` as simple resource controller interface

It is common that there are multiple users who use the same HPC cluster. Moreover, in many cases, these users submit their jobs to central resource manager by specifying necessary resources for their jobs. One example is introduced in Chapter 2 [31]. In such clusters, one simple solution of managing nodes dynamically is to turn-off non-reserved nodes. When the users need to run their jobs on certain resources, an interface between to power controller and them is required. `cntools` suits this purpose very well. `cntools` gets the resource request from system users and keeps track of who reserved what resources what time. When someone has already requested a node, it does not allow others to request that resource. Only the requested user of a resource or the administrator of the system can release that resource. Therefore, it avoids certain inconsistencies induced by mismanagement of resources. If certain resources are not requested ClusterNap keeps them OFF.

5.1.1 Turning ON and OFF servers and devices when there are dependencies

While there are many similar power controller interfaces none of them considers the dependency of nodes in the system. To check the usability of `cntools` and dependency-awareness of ClusterNap, we did experiment on the HPC cluster, Cloko. The Cloko consists of several filesystem servers, RAIDs and switches which are connected to hundreds of compute nodes. The main structure of the cluster is similar to the cluster structure illustrated in

the Figure 1.1. In this cluster, for example, to use a certain compute node, the corresponding filesystem server, RAID and switches must be ON beforehand (RUN-dependency). To turn the filesystem server ON, we need to turn the corresponding network switches and RAID beforehand. To check if ClusterNap works on this dependency, we chose three compute and a filesystem (Dell PowerEdge R815) servers. Moreover, a switch for management (Dell PowerConnect 5448), a switch for compute (Summit X650), and a RAID (VCRVAX) were used. The dependencies are shown in the Figure 5.1.

From the power state where only the management node and the management switch being ON (leftmost state in Figure 5.1), we requested the compute node `cloko121`. In the Figure 5.1, each directed line shows a dependency. The red ones are ON, the blue ones are OFF and the black directed lines show RUN-dependencies. According to this dependency, to turn the compute node `cloko121`, we must turn one the compute switch and the RAID first. After these two nodes become ON, we then must turn the fileserver node ON. Finally, it is possible to turn the compute node ON.

We have defined the nodes and their dependencies in ClusterNap. For each node, we also have made simple ON/OFF scripts which ClusterNap executes when necessary. Since ClusterNap is dependency aware, only thing a user need to do is just request the node by `cntools`. The nodes' states are updated by Nagios (using Nagios as node state updater module is explained in the Chapter A.3.2).

After running the command "`cntools request cloko121`", ClusterNap gracefully turned each node ON in the correct order. In 13 minutes, from the all but management nodes being OFF, our requested node became ON and we could access to and use that node (see Figure 5.1).

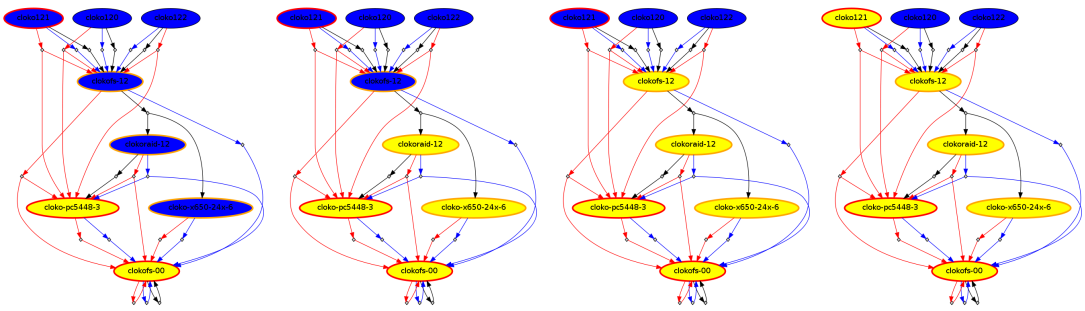

Figure 5.1. Turning-on a compute node, starting from switch, RAID, and filesystem server took 13 minutes.

After this demonstration, we *released* the node, `cloko121`, we previously requested. The `cloko121` is no longer in need, so are the other nodes such as corresponding filesystem server and switches. Therefore, ClusterNap should turn-off them in the reverse order.

This time it took 6 minutes to reach the minimum-power state gracefully from the initial state as shown in the Figure 5.2. The main aim of this demonstration is the gracefulness of state transition rather than the time measured because transition time differs for different resources.

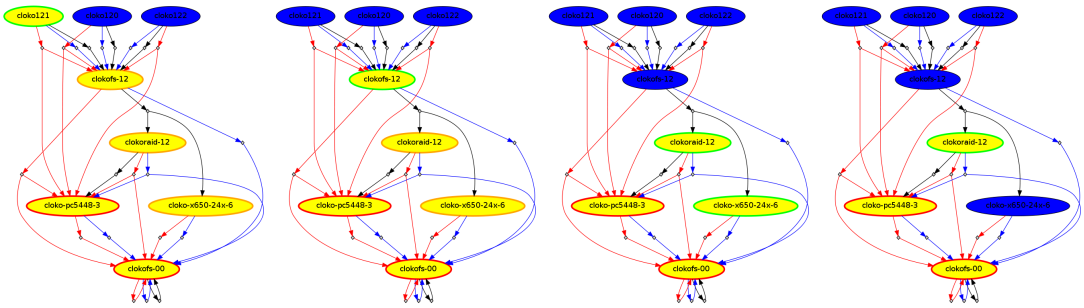

Figure 5.2. Turning-off a compute node, a filesystem server, a RAID and a switch took 6 minutes.

5.2 Control multiple sites from remotely as administrative use

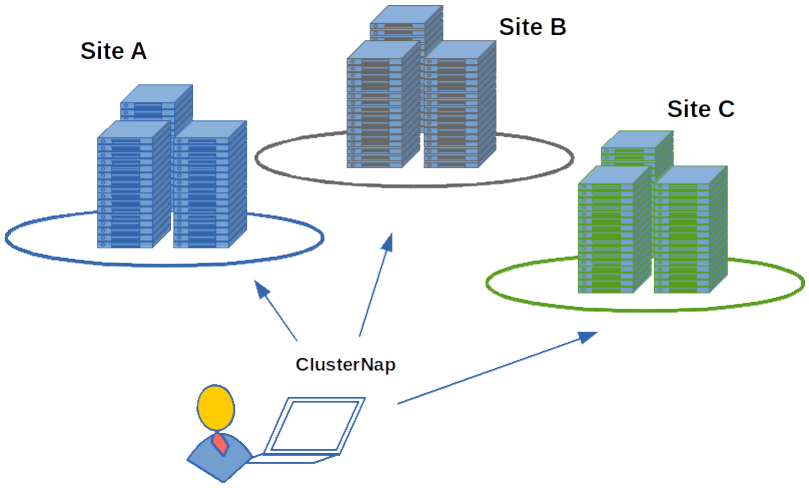

Figure 5.3. Controlling multiple clusters and resources remotely

Another possible application of ClusterNap is administrative use. As long as the necessary ssh connection and security permissions are given, ClusterNap can operate from the remote server as shown in the Figure 5.3. This feature allows it to monitor multiple HPC clusters simultaneously for the administrative purpose. We, for example, set two clusters – Cloko and Hongo – on ClusterNap running on the remote node. We successfully controlled these two sites from the single interface of ClusterNap. We also turned-ON and OFF 77 physical servers on Cloko simultaneously by using cntools. Without any apparent problem, the system reached the desired state in less than 10 minutes.

5.2.1 Example: Cloko, InTrigger, EC2 instances

The nodes defined in ClusterNap do not need to be physical only. We can also configure and control virtual machines and cloud instances as well. In this scenario, we demonstrates

that ClusterNap can control cloud instances. we have implemented state changing script for Amazon's EC2 instance by using the API provided by Amazon. In addition, we update their states by Nagios by implementing simple state changing scripts (for more information on how we configure the instances, see the Section A.2).

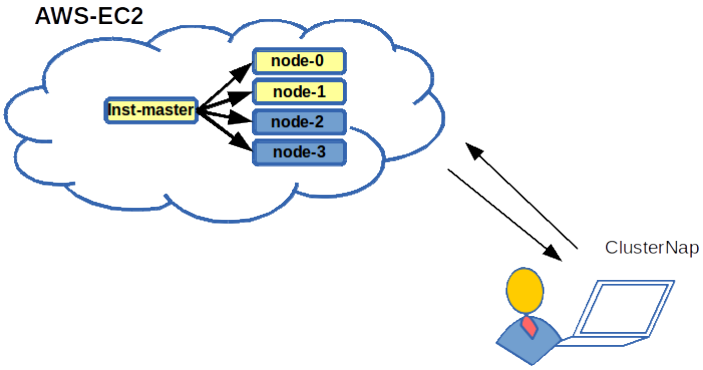

Figure 5.4. Controlling the state of cloud instances remotely

By providing the instance's ID, geographical region name, EC2's access key, and secret key as arguments to the script we have written, we could successfully control their states form ClusterNap's interface remotely as illustrated in the Figure 5.4.

5.3 On the cloud

We demonstrate that it is trivial to use ClusterNap on the cluster in the cloud. We have created a virtual cluster by setting up EC2 instances. On the master node we have simply installed ClusterNap as if we do it in physical cluster. Just creating the same configuration we have done when we control the cloud instances from remote node, ClusterNap controls its instances from the master node as illustrated in the Figure 5.5. In this way, we can use ClusterNap in cloud as resource managing interface as we do in physical clusters explained in the Section 5.1.1.

Example: ec2 + cntools + qsub

One frugal way of using instances of cluster in the cloud is to turn-on the only necessary resources when only needed, and turn them off otherwise. However doing so is somewhat tiresome. In case of Amazon's EC2 environment, for instance, a user has to log-in to the web interface and turn-on the necessary nodes one by one by clicking each instances. However, giving everyone the access to the web interface is not a good idea in terms of security.

ClusterNap also works with Torque batch scheduler. By using the `cntool`'s `qsub` option in this situation, we can avoid activities such as manually turning ON/OFF nodes. When the all worker nodes in Figure 5.5 were OFF, we run following command,

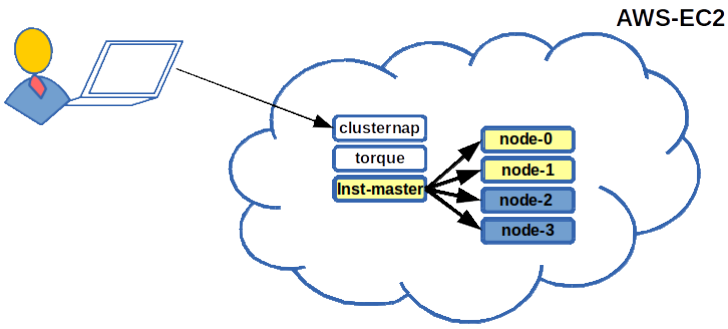

Figure 5.5. Controlling the state of cloud instances by integrating resource management application.

```
echo ``sleep 30'' | cntools qsub -l nodes=node002
```

ClusterNap in this situation, first identifies the requested nodes from the arguments and standard input of qsub, then turns them ON if they are not requested and OFF. Once the necessary resource is ready, Torque's scheduler *pbs_sched* executes gets the information that the necessary resource has become available and run the submitted job in queue. After the completion of the submitted job, ClusterNap identifies the completion and shuts down the resources. In the command shown above, all these phases – turn-on, execute, turn-off – are completed in 100 seconds. 50 seconds after submitting the job, the required resource has become available. Then, *pbs_sched* recognizes the requested node `node002` has become available and run the job (to sleep 30 seconds). The job has completed in 82th second since the submission. Finally, in the 102th second the used node `node002` has become OFF again. All of above phases are completed automatically and we can run the same command in physical resources as well, while only the state transition time might become longer for physical servers.

The point of this experiment is that a user can use a cluster in the cloud without even bothering about the current states of the resources since ClusterNap takes care of it. Only thing the user should do is specify the required resource and submit the job through `cntools qsub`.

5.4 Integration with other applications: GXP make workflow

In Section 4.3 we introduced ClusterNap's simple API. Along with the user interface `cntools`, one can integrate her applications by using its API so that one system can dynamically provide its resources for both users and their applications consistently at the same time.

To show a simple example, we used ClusterNap's API with GXP make [34], a workflow system which makes full use of GNU make and extends its use to distributed systems. By

using `-j` option of GNU make, one can execute possible executions in parallel. GXP make basically grabs those tasks and executes them in the distributed environment.

When one runs a large workflow work in such workflow systems, the number of tasks that can be executed in parallel changes over time. For example, at one point of execution there might be thousands of tasks can be executed in parallel, but at another point that number reduce to less than ten and continues in that situation very long time. In this scenario, when the number of tasks executable in parallel is too few that the rest of the resources which are not executing any task just become idle. It means those resources are just wasting energy/cost. Therefore, our final ambition is to create a platform which activates and deactivates such idle resources by using GXP make's scheduler and ClusterNap's API according to the number of the tasks executable in parallel during runtime.

While we have not achieved such platform by using GXP make yet, we implemented very basic functionality. In this experiment, we executed the simplest GXP make workflow job. It just creates given number, `N`, of tasks. Each task is just a sleep function. Therefore, the scheduler of GXP make sends the task to certain node, that task sleeps there for certain time, in our case it is 100 seconds.

We use a small cluster on cloud which consists of four worker nodes, and a master node. The number of the tasks is 50. First, we run the this workflow when all nodes are ON. As predicted, the result is as shown in the Figure 5.6. The vertical axis shows the number of tasks being executed in parallel and the horizontal axis represents the execution time. All task were executed in all resources one after another in parallel. Then, we executed the same workflow when all the workers are OFF. Our simple function then gets the task numbers from the GXP make's scheduler, and turns ON necessary resources by using ClusterNap's API. The result is shown in the Figure 5.7.

As seen in the graph, all possible tasks are not executed immediately. Nodes become ON one by one and as a result the workflow is finished 50 seconds later than the previous situation, where all nodes are ON from the beginning. After the completion of the workflow, all nodes are turned OFF automatically since there is no task left.

The main point of this demonstration is not the performance or cost efficiency in this case (which however are the final destination), but is the demonstration of the usability of ClusterNap's API with integration of other applications such as workflow systems. The important usage of such integration is when the number of tasks changes greatly over long period of time. Currently we are looking forward to implement such dynamic resource provisioned workflow system.

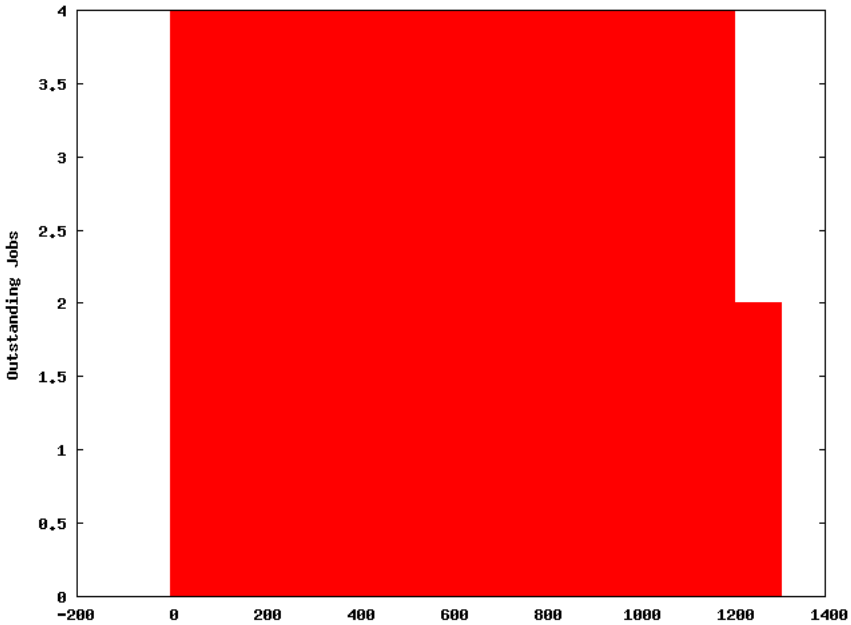

Figure 5.6. The number of tasks being executed in parallel without the task number aware functionality

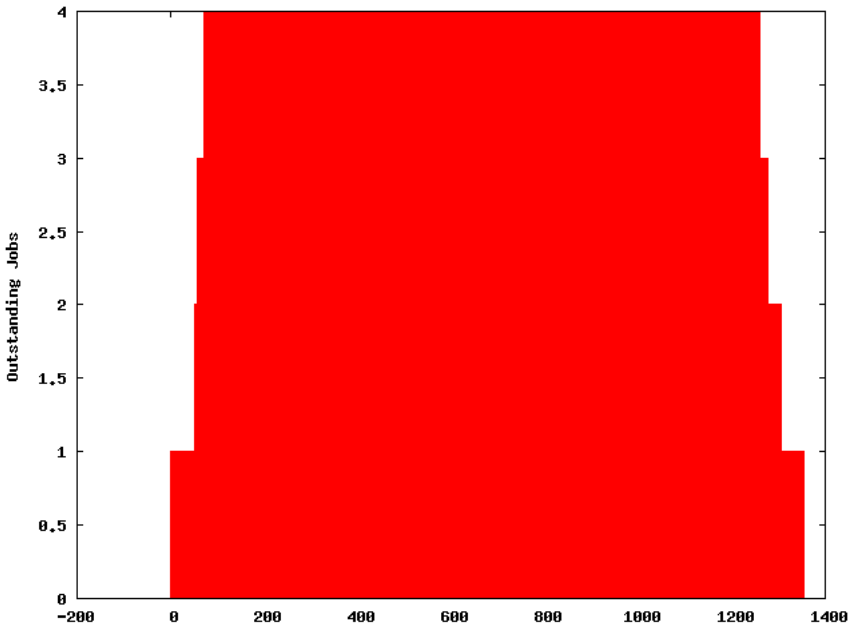

Figure 5.7. The number of tasks being executed in parallel with the task number aware functionality

Chapter 6

Conclusion

We propose a resource dependency aware, generic node power controller tool, ClusterNap. ClusterNap is generic enough to control states of various kinds of resources such as physical servers, network switches, and RAIDs. Moreover, it can also be used for controlling virtual resources such as VMs and cloud instances. To our best knowledge, the dependency-awareness of ClusterNap is the one feature that no other similar tools have yet offered.

In this thesis, we introduced the Power State Transition Algorithm and ClusterNap which implements the algorithm. In addition, several application scenarios of ClusterNap are introduced as an evaluation of the tool. We showed application scenarios in physical clusters, and cloud environments, controlling both from local and remote nodes. Usability of its API is also introduced. Even though it does not have an independent resource state checking module yet, we showed how we can use other resource managing and monitoring tools (i.e., Nagios and Torque) to update the resource states.

Future works

At the moment, ClusterNap does not have its own resource state checking module yet. It is, however, the very next feature we are looking forward to implement. Furthermore, the optimization of the minimum power state function should be improved along with the node configuration syntax. We are planning to add more features to the node configuration syntax, such as the type of nodes (e.g., physical host, VM, service, and cloud instance).

To achieve the best result in energy-efficiency, excellent policy is probably the most important factor along with other facets. Therefore, easy and efficient policy making module in the near future has high priority.

Acknowledgements

First of all, I want to express my gratitude to my professor Kenjiro Taura for accepting me as a member of his group in both undergraduate and masters courses. During these three years, he has taught me countless things through his great lectures, comments in meetings, and even through mere conversations. His way of thinking to find and approach any problem in real life is brilliant and I always admire it. Everyday of these years became a learning new things from different view and most importantly I gained a motivation to fail and learn new things thanks to him.

I also thank my parents and family members. Without their constant support to this day I could not do a bit of my research. My girl-friend and other friends, I cannot imagine my university life without them. You are the ones who put color during these years.

I also owe an appreciation to the people of NS-solutions group. Without their scholarship, last two years of my study were impossible. Accepting me as a scholarship student not only helped financially, it also gave me confidence to continue my graduate course and my research.

Yokoyama Daisaku san was always there to help me when I need any help on my research. Without his kind assistance, I could not even graduate my bachelors course. Even during my masters course, he always encouraged me to continue my research and pointed out the mistakes. At this very moment, he is one of the very few people who really understand what is going on behind the algorithm proposed in this paper.

Finally, my lab colleagues, Hayashi, Nakatani, Nakazawa, senpais and kouhais, you are the ones who made this incredible memories of my student life. Let us remain in touch even after our graduation.

Bibliography

[1] Adaptive computing, 2014. Available at `http://www.adaptivecomputing.com/`.

[2] Eucalyptus, 2014. Available at `http://www.eucalyptus.com`.

[3] intrigger, 2014. Available at `www.intrigger.jp/`.

[4] Opennebula, 2014. Available at `http://www.opennebula.org`.

[5] Openstack, 2014. Available at `http://www.openstack.org`.

[6] Luiz André Barroso and Urs Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[7] Leander Beernaert, Miguel Matos, Ricardo Vilaça, and Rui Oliveira. Automatic elasticity in openstack. In *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*, page 2. ACM, 2012.

[8] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755–768, 2012.

[9] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, et al. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[10] Rajkumar Buyya, Anton Beloglazov, and Jemal Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308*, 2010.

[11] Fábio Coutinho, Luís Alfredo V de Carvalho, and Renato Santana. A workflow scheduling algorithm for optimizing energy-efficient grid resources usage. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 642–649. IEEE, 2011.

[12] DELL. Dell dell remote access card for remote server, 2014. Available at `http://www.dell.com/content/topics/global.aspx/power/en/ps2q02_bell?c=us`.

[13] Amgalan Ganbat. Clusternap, 2014. Available at `https://github.com/amgaa/ClusterNap`.

[14] Developer Guide. Amazon auto scaling. 2010.

[15] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. Reducing disk power consumption in servers with drpm. *Computer*, 36(12):59–66, 2003.

[16] Sijin He, Li Guo, Yike Guo, Chao Wu, Moustafa Ghanem, and Rui Han. Elastic application container: A lightweight approach for cloud resource provisioning. In *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, pages 15–22. IEEE, 2012.

[17] HP. Hp integrated lights out (ilo), 2014. Available at `http://h17007.www1.hp.com/us/en/enterprise/servers/management/ilo/index.aspx`.

[18] Liting Hu, Hai Jin, Xiaofei Liao, Xianjie Xiong, and Haikun Liu. Magnet: A novel scheduling policy for power reduction in cluster with virtual machines. In *Cluster Computing, 2008 IEEE International Conference on*, pages 13–22. IEEE, 2008.

[19] IBM. Ibm endpoint manager for power management, 2014. Available at `http://www-03.ibm.com/software/products/en/ibmendpmanaforpowemana/`.

[20] Emir Imamagic and Dobrisa Dobrenic. Grid infrastructure monitoring system based on nagios. In *Proceedings of the 2007 workshop on Grid monitoring*, pages 23–28. ACM, 2007.

[21] J.R.Arredondo. Rackspace auto scale, 2013. Available at `http://www.rackspace.com/blog/start-using-auto-scale-today`.

[22] Simon Kiertscher, Jorg Zinke, Stefan Gasterstadt, and Bettina Schnor. Cherub: power consumption aware cluster resource management. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 325–331. IEEE Computer Society, 2010.

[23] Jinoh Kim, Jerry Chou, and Doron Rotem. Energy proportionality and performance in data parallel computing clusters. In *Scientific and Statistical Database Management*, pages 414–431. Springer, 2011.

[24] Jinoh Kim and Doron Rotem. Energy proportionality for disk storage using replication. In *Proceedings of the 14th International Conference on Extending Database Technology*, pages 81–92. ACM, 2011.

[25] Kyong Hoon Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of cloud resources for real-time services. In *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, page 1. ACM, 2009.

[26] Krzysztof Kurowski, Ariel Oleksiak, Michal Witkowski, and Jarek Nabrzyski. Distributed power management and control system for sustainable computing environments. In *Green Computing Conference, 2010 International*, pages 365–372. IEEE, 2010.

[27] Nitesh Maheshwari, Radheshyam Nanduri, and Vasudeva Varma. Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework. *Future Generation Computer Systems*, 28(1):119–127, 2012.

[28] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12. IEEE, 2011.

[29] Ming Mao, Jie Li, and Marty Humphrey. Cloud auto-scaling with deadline and budget constraints. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 41–48. IEEE, 2010.

[30] Shuangcheng Niu, Jidong Zhai, Xiaosong Ma, Xiongchao Tang, and Wenguang Chen. Cost-effective cloud hpc resource provisioning by building semi-elastic virtual clusters. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, page 56. ACM, 2013.

[31] A-C Orgerie, Laurent Lefèvre, and J-P Gelas. Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In *Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on*, pages 171–178. IEEE, 2008.

[32] Ronald L Rivest and Charles E Leiserson. Introduction to algorithms. 1990.

[33] Garrick Staples. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 8. ACM, 2006.

[34] Kenjiro Taura, Takuya Matsuzaki, Makoto Miwa, Yoshikazu Kamoshida, Daisaku Yokoyama, Nan Dun, Takeshi Shibata, Choi Sung Jun, and Jun 'ichi Tsujii. Design and implementation of gxp make—a workflow system based on make. *Future Generation Computer Systems*, 29(2):662–672, 2013.

[35] Tiffany Trader. Green power management deep dive, 2013. Available at `http://www.greencomputingreport.com/gcr/2013-06-05/green_power_management_deep_dive.html`.

[36] Chris Tucker, David Shuffelton, Ranjit Jhala, and Sorin Lerner. Opium: Optimal package install/uninstall manager. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 178–188. IEEE, 2007.

[37] Chao-Tung Yang, Kuan-Chieh Wang, Hsiang-Yao Cheng, Cheng-Ta Kuo, and William Cheng-Chung Chu. Green power management with dynamic resource allocation for cloud virtual machines. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 726–733. IEEE, 2011.

[38] Andrew J Younge, Gregor Von Laszewski, Lizhe Wang, Sonia Lopez-Alarcon, and Warren Carithers. Efficient resource management for cloud computing environments. In *Green Computing Conference, 2010 International*, pages 357–364. IEEE, 2010.

Appendix A

ClusterNap

A.1 Getting and installing ClusterNap

Installing ClusterNap is as simple as downloading the package and adding necessary scripts to environment path.

```
$ git clone https://github.com/amgaa/ClusterNap
$ ln -s /path/to/ClusterNap/bin/clusternapd /usr/local/bin/clusternapd
$ ln -s /path/to/ClusterNap/bin/cntools /usr/local/bin/cntools
```

Alternatively, we can add the `Clusternap/bin/` directory to the system's path. After configuring nodes, we start ClusterNap by starting the daemon as follows.

```
$ clusternapd start
```

A.2 Configuring nodes

Before starting the daemon, `clusternapd`, we should add the nodes which we want to control to the ClusterNap. All configuration files should be in `/path/to/ClusterNap/config/` directory and should have `.conf` extension.

For example, let's create configuration files for the servers, raids, switches, and EC2 instances.

```
$ cd /path/to/ClusterNap/config/
$ touch servers.conf
$ touch raids-switches.conf
$ touch ec2-instances.conf
```

A.2.1 Servers

Following is an example of configuring servers in the file `servers.conf` we created.

```
define node{
  name:             localhost
  run_dependencies: localhost
  on_dependencies:  localhost
  off_dependencies: localhost
  on_command:       localhost, root, None
  off_command:      localhost, root, None
}

define node{
```

```
  name:              master
  run_dependencies: raid_A, switch_A
  on_dependencies:  localhost
  off_dependencies: localhost
  on_command:       localhost, root, ipmi!10.0.0.101
  off_command:      localhost, root, ssh_shutdown!192.168.101
}

define node{
  name:              worker[001-100]
  run_dependencies: master
  on_dependencies:  localhost | master
  off_dependencies: localhost | master
  on_command:       localhost, root, ipmi!10.0.0.[1-100] |\
                    master,    root, ipmi!10.0.0.[1-100]
  off_command:      localhost, root, ssh_shutdown!192.168.0.[1-100] | \
                    master,    root, ssh_shutdown!192.168.0.[1-100]
}

define_command {
  name:              ssh_shutdown
  command_line:     ssh $ARG1 shutdown -h now
}

define command {
  name:              ipmi
  command_line:     ipmitool -I lanplus -H $ARG1 -P ipmi_password -U root chassis power on
}

define command {
  name:              None
  command_line:     echo ``DO NOTHNING''
}
```

A.2.2 RAIDs, switches

Following is an example of configuring servers in the file `raids-switches.conf`.

```
define node{
  name:              raid_A
  run_dependencies: switch_A
  on_dependencies:  localhost
  off_dependencies: localhost
  on_command:       localhost, root, snmp_on!snmp_ip!snmp_port_no
  off_command:      localhost, root, snmp_off!snmp_ip!snmp_port_no
}


define node{
  name:              switch_A
  run_dependencies:
  on_dependencies:  localhost
  off_dependencies: localhost
  on_command:       localhost, root, snmp_on!snmp_ip!snmp_port_no
  off_command:      localhost, root, snmp_off!snmp_ip!snmp_port_no
}

define command {
  name:              snmp_off
  command_line:     snmpset -c comm_str -v 2c $ARG1 .1.3.6.1.4.1.13742.4.1.2.2.1.3.$ARG2 i 0
}

define_command {
  name:              snmp_on
  command_line:     snmpset -c comm_str -v 2c $ARG1 .1.3.6.1.4.1.13742.4.1.2.2.1.3.$ARG2 i 1
}
```

A.2.3 EC2 instances

Following is an example of configuring Amazon's in the file `ec2-instances.conf` we created.

```
define node{
  name:                ec2-instant
  run_dependencies:    localhost
  on_dependencies:     localhost
  off_dependencies:    localhost
  on_command:          localhost, amgaa, ctl_ec2_inst\
                       !ap-northeast-1 (or other region)\
                       !EC2-ACCESS-KEY \
                       !EC2-SECRET-KEY \
                       !instant-ID \
                       !on
  off_command:         localhost, amgaa, ctl_ec2_inst \
                       !ap-northeast-1 (or other region)\
                       !EC2-ACCESS-KEY \
                       !EC2-SECRET-KEY \
                       !instant-ID \
                       !off
}

define command{
  name:           ctl_ec2_inst
  command_line:  /path/to/ClusterNap/nagios-plugins/ctl_ec2_instance.py \
                                          $ARG1 $ARG2 $ARG3 $ARG4 $ARG5
}
```

A.2.4 Other resources

ClusterNap is a generic tool to control resources. Therefore, its usage is not limited by servers, raids, switches, or cloud instances. The type of resources and go on to anything that a user can imagine. For example, user can define a certain service as a node, and configure its ON/OFF (start/stop) commands and dependencies so that ClusterNap start and stop that node as user requests. A resource can even be component of a computer such as CPU.

A.3 Setting up the state checking module

To work properly, the power states of nodes must be updated constantly and correctly. The power state of each node is represented a file in the directory
`/path/to/ClusterNap/state/nodes/`

There are files generated automatically with the same name of each nodes defined in the `config/` directory. Inside these files in the `state/nodes/` directory, either `1`, `0`, or `-1` should be written. When the node is ON, it should be `1`, when OFF `0`, when UNKNOWN `-1` should be written inside these files. To update nodes' states means, therefore, to update these files. Currently, we use Nagios monitoring system or Torque resource manager to update these files.

A.3.1 Using Torque

When the node names match with the host names used in Torque resource manager, we can use Torque as node state updater. ClusterNap uses Torque's `pbsnodes` command and updates its nodes' statuses. The only requirement is that the names of the nodes defined in ClusterNap and the Torque must match with each other. To use Torque, we should just set environment variable `CHECK_TORQUE` to `True` when we start `clusternapd`. Therefore, it should be as simple as executing following command:

```
$ CHECK_TORQUE=True clusternapd start
```

A.3.2 Using Nagios

When the user uses Nagios resource monitoring system, it is also good idea to take the advantage of its scalable resource checking engine. Nagios uses either its default or user defined plugins to check its resources. Just by changing those plugins or using the plugins we provide (in the directory `/path/to/ClusterNap/nagios-plugins/`), we can make Nagios to update ClusterNap's nodes' statuses. Following is an example of how we use ClusterNap's plugins in Nagios' config file (Remember it is **NOT** ClusterNap's config file, but Nagios'. Nagios3's config files, for example, reside in `/etc/nagios3/conf.d/` directory by default).

```
define command{
        command_name            check_host_CN
        command_line            /path/to/ClusterNap/nagios-plugins/check_ping_CN.py\
                                                                -H $HOSTADDRESS$ \
                                                                -w 5000,100%     \
                                                                -c 5000,100%     \
                                                                -p 1
        }

define command{
        command_name            check_ec2_instance
        command_line            /path/to/ClusterNap/nagios-plugins/check_ec2_instance_CN.py \
                                                                        $ARG1$ $ARG2$ $ARG3$ \
                                                                            $ARG4$ -H $ARG5$
        }

define host{
        use                     generic-host
        host_name               worker001
        alias                   worker001
        address                 192.168.0.1
        check_command           check_host_CN
        }

define host{
        use                     generic-host
        host_name               ec2-instant
        alias                   ec2-instant
        address                 0
        check_command           check_ec2_instance!ap-northeast-1!\
                                        EC2-ACCESS-KEY!\
                                        EC2-SECRET-KEY!\
                                        instant-ID!\
                                        ec2-instant
        }
```

A.4 Using the API

ClusterNap has API for python. Following is an example of using the API,

```
$ python
>>> import cntools
>>> cn = cntools.clusternap()
>>> cn.request('nodeA')
>>> cn.release('nodeB')
...
```

To see the more information, about the API functions

```
>>> help(cn)
```

The `help(cn)` will display the API functions and their roles.

A.5 Troubleshooting and logging

When there is a trouble, best way to track the problem is to see logs. ClusterNap records all logs in `/path/to/ClusterNap/logs/` directory. There are two types of log files in this directory. First one is *event* log which has the name in `YYYY-MM-DD_event.log` format. Another one is *error* log which has the name in `YYYY-MM-DD_error.log`. Event log records every action ClusterNap takes such as sending ON/OFF commands, releasing/requesting nodes. On the other hand, Error log records errors such as configuration error, unknown node request error. One can easily see the ClusterNap action from the log files using `tail` command in real time. To track the events, for instance, one could do following:

```
$ tail -f /path/to/ClusterNap/logs/2014-05-15_event.log
```

and to track error logs,

```
$ tail -f /path/to/ClusterNap/logs/2014-05-15_error.log
```

Appendix B

Greedy Set Cover Algorithm

Problem definition

Input:

$$U = e_1, e_2, ..., e_n$$
$$S = S_1, S_2, ..., S_k$$
$$c : S \to \mathbb{Z}^+$$

Output:

$$\min_{S'} c(S') = \min_{S'} \sum_{S \in S'} c(S)$$

subject to

$$\forall i,\ \exists j \quad such\ \ that \quad S_j \in S'\ and\ e_i \in S_j$$

The algorithm

Algorithm B.1 Greedy Minimum Set-Cover Algorithm

```
Input:
Element set U = e_1, e_2, ..., e_n,
subset S = S_1, S_2, ..., S_k,
cost function c : S → Z^+
Output: Set cover C with minimum cost
C ← ∅
while C is not a set cover do
    Pick a set S with the minimum cost
    C ← C ∪ S
end while
return C
```