A Comparative Study of Baremetal Provisioning Frameworks

Ashok Chandrasekar, Garth Gibson

CMU-PDL-14-109
December 2014

Parallel Data Laboratory
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

There are many baremetal frameworks available in market today. These baremetal frameworks help ease the work of a cluster/datacenter administrator by automating the deployment of operating systems and other software onto the individual machines in a datacenter. These frameworks differ from each other on several aspects like price, stability, maintainability, feature support and performance. As a result, it becomes difficult to choose the best framework which suits one's needs. This study aims at helping the administrator in choosing the appropriate framework based on their needs by providing a comparison of these frameworks with a main emphasis on Emulab and OpenStack Ironic.

Acknowledgements: This material is supported in part by the National Science Foundation under awards CNS-1042537 and CNS-1042543 (PRObE), by Intel as part of the Intel Science and Technology Center for Cloud Computing (ISTC-CC), by gifts from Yahoo!, APC, EMC, Facebook, Fusion-IO, Google, Hewlett-Packard, Hitachi, Huawei, IBM, Intel, Microsoft, NEC, NetApp, Oracle, Panasas, Riverbed, Samsung, Seagate, STEC, Symantec, and VMware. We thank the member companies of the PDL Consortium for their interest, insights, feedback, and support.

Keywords: baremetal frameworks, network simulators, scheduling, simulated annealing, emulab, ironic

1 Introduction

Any hardware associated with computers require a software installed on top of it and configured properly before it can function. For example, a normal desktop or laptop computer requires an operating system setup and appropriate drivers configured before it can be used. This installation and configuration is usually time consuming even in the case of a single machine. Now consider a cluster of 100 or 1000 machines of bare hardware which needs to be setup with an OS and software. This can take hours of manual work on the part of a system administrator to complete. Efforts have been made to automate this process. As a result there are many baremetal provisioning tools in market today, which are being widely used in setting up clusters both in academia and in industry.

One of the key challenges of a baremetal provisioner is scheduling the appropriate nodes from the cluster. Scheduling is very hard to optimize mainly because of the number of variables involved in making a decision. As a result it is considered to be a NP hard problem [1, 2]. Because of this there is no single perfect way to schedule resources, but a set of strategies can help mitigate the cost involved in scheduling. Baremetal scheduling is not as difficult as scheduling virtual machines, which involves co-location, migration, etc. Rather baremetal scheduling is relatively simpler because of the one to one mapping of the physical machines to the machines that need to be scheduled. But there are other issues like difference in hardware in terms of amount of resources, manufacturers, etc. which can lead to difficulty in scheduling.

There are various other challenges involved in baremetal provisioning like networking, support for different hardware, etc. This offers us a wide variety of tools to choose from for setting up our cluster. The range of choices also leads to confusion among people as to which will be the most appropriate tool for their particular cluster setup. This report is a survey of some of the popular baremetal provisioning frameworks like Emulab [3], Ironic [4], Crowbar [5], Maas [6], Cobbler [7] and Razor [8] mainly aimed at helping people to choose the apropriate framework for their cluster.

2 Baremetal Provisioning

A typical baremetal provisioner follows the following steps when provisioning a single node or a set of nodes as mentioned in fig 1: First, the available hardware has to be registered with the baremetal provisioner which is hosted on one of the nodes on the cluster. This registration can either be manual by specifying the physical machine's MAC address to the provisioner or automatically discovered by the baremetal provisioner. When a request for node allocation comes, the provisioner picks the appropriate set of nodes from the available list that it maintains. This selection is based on the scheduling algorithm used by the provisioner. This scheduling algorithm or policy varies across schedulers. After selection, the provisioner switches on the set of nodes using a power driver like IPMI (Intelligent Power Management Interface) configured on the physical machines. Once the nodes are switched on, they start to boot and wait for the DHCP/PXE (Preboot Execution Environment) server running on the provisioner to discover, and download a bootstrap image from the provisioner using TFTP (Trivial File Transfer Protocol). Once this bootstrap image is sent to the guest nodes, they start downloading the actual OS image specified by the user. Once the download is complete, the guest node restarts again but this time boots from the OS image on the hard disk. To complete the provisioning process, the guest node replies back to the baremetal provisioner saying it is up and running. This process is common to all the baremetal provisioning frameworks under study. But there will be minor variations in the tools' handling of each of these separate steps.

In most cases, the baremetal provisioners are used along with a Configuration Management System (CMS). This is because just installing the operating system is not enough in a lot of cases. Consider setting up a Hadoop [9] baremetal cluster, which involves installing Hadoop on each of the machines after the installation of OS. This installation involves a lot of configuration which once again takes time and can be better if automated. To automate this process there are a lot of CMS tools available like Opscode Chef [10], PuppetLabs Puppet [11], Juju [12], etc. These tools depend on the baremetal provisioner used. So it is important that we choose the appropriate baremetal provisioner.

3 Baremetal Frameworks

This section will briefly discuss the frameworks that are chosen for this study.

3.1 Emulab

Emulab [3] is a network testbed which provides an environment to carry out research in computer networks and distributed systems by allowing user to provision bare hardware. It requires Emulab to be pre-installed in the cluster. To provision nodes, Emulab takes a network topology specified by the user in a Network Simulator (NS) file, and allocates the required number of nodes and connects them using the mentioned topology. Users can then use this as a small private cluster.

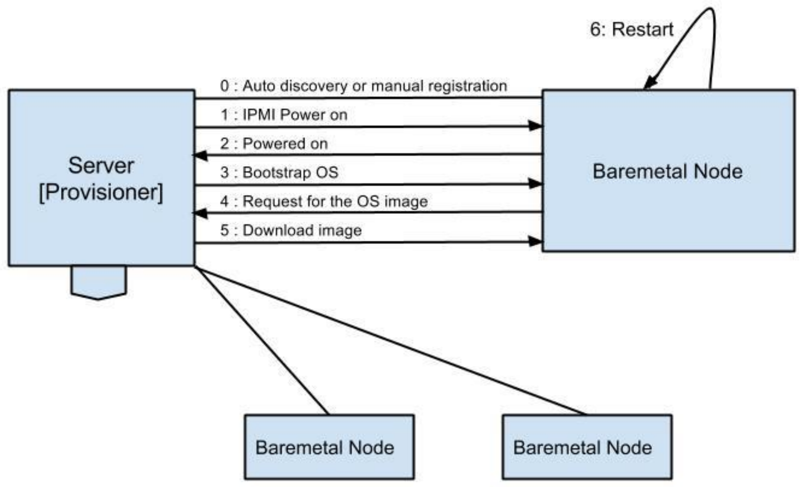

Figure 1: Default workflow in baremetal provisioning

3.2 OpenStack Ironic

Ironic [4] is OpenStack's new baremetal provisioning framework. It was first released in the latest Icehouse release of OpenStack [13]. Ironic has spanned into a separate project inside OpenStack which was initially tied to Nova (OpenStack's compute cloud). Ironic helps user to provision bare hardware and launch Ubuntu and other Linux based operating systems. The main rationale behind Ironic's inception is to make OpenStack self-sufficient and make it a single platform which can launch both baremetal and virtual machines. It depends on other OpenStack components like nova, glance, keystone, etc. It is also integrated with an orchestration layer called Heat, which helps in configuring various components of an OpenStack baremetal cloud.

3.3 Dell Crowbar

Dell Crowbar is a baremetal provisioning framework developed by Dell. The Crowbar project was aimed at making deployment of OpenStack cloud on top of a cluster of bare hardware simple. Dell Crowbar can be used to provision baremetal nodes, and setup OpenStack cloud or Apache Hadoop on top of it. To deploy these services like OpenStack cloud and Hadoop, Crowbar uses configuration management modules known as barclamps which automates this process. Crowbar's version 1 is maintained by Dell. The latest release of crowbar (version 2) called OpenCrowbar is open source and community maintained [14].

3.4 Cobbler

Cobbler is another open source framework for provisioning bare hardware. The Cobbler project was initiated by RedHat and now functions independently. But the main focus of deployment is still on Fedora and RedHat related Linux systems. Cobbler has its own CMS and also supports Chef to automate the deployment of tools.

3.5 Canonical MaaS

Canonical MaaS (Metal as a Service) is an Ubuntu offering for baremetal provisioning. It helps in deploying Ubuntu onto multiple baremetal machines. Like the other baremetal provisioners, MaaS is tightly coupled with juju which acts as its configuration management system.

3.6 Razor

Razor is a baremetal provisioning framework built by Puppet Labs. PuppetLabs had a CMS known as Puppet which can help configure multiple machines simultaneously. Razor was built to complement Puppet so that

PuppetLabs' solution becomes a complete package from deploying a cluster to configuring it with the necessary software and tools.

4 Evaluation

Evaluation of the aforementioned baremetal provisioning frameworks has been done both qualitatively and quantitatively. The qualitative analysis is really important as there are a lot of variables in choosing a baremetal provisioning framework which when not considered properly can lead to difficulty during setup. Also once setup, it is difficult to migrate to a different framework as all the setup has to be done from the beginning using the new framework and some framework specific features like migration or preemption can lead to problems with the already alive deployments. All the baremetal provisioning frameworks mentioned in the previous section are evaluated on the following criteria.

- Vendor lock-in
- Maturity of the project
- Number of deployments/users
- Difficulty of installation
- Stability
- Difficulty of maintenance
- Hardware requirements
- Breadth of features
- Scheduling

We chose the criteria mentioned above for this study because these criteria represent some of the most important components of any software like stability, maintenance, installation, etc.

4.1 Vendor Lock-in

Vendor lock-in is one of the important criteria which will help in choosing a framework. This is because the vendor's pricing, support and flexibility will directly affect the usage and maintenance of the framework. It also directly affects the monetary cost involved in setup, maintenance and change of framework and hardware.

All the baremetal provisioning frameworks under study are open source or at least has an equivalent open source version. OpenStack Ironic, Emulab, Cobbler and MaaS are completely open source and free. Whereas Dell's Crowbar has an open source equivalent of the enterprise solution which comes as a package with Dell hardware, crowbar and an OpenStack or Cloudera's distribution of Hadoop setups. The University of Alabama at Birmingham is one of the popular academia users of the Dell's enterprise package of Crowbar [15]. The open source version of Crowbar is free and is being actively contributed. So there is no actual lock-in or other disadvantage associated with the open source version. Puppet Lab's Razor like Crowbar has an enterprise version which offers more features and support. But just like Dell, Razor has an open source version which is being actively contributed.

4.2 Maturity of the project

Maturity of the project is important because it directly correlates to issues that are a result of untested or immature code which might break. So it is important to ensure that a particular baremetal provisioning framework is mature and usable. In this study, maturity is computed mainly by the number of years that the project has been active. This does not assure the most complete evaluation of maturity, as it also depends on the developers involved in the project and the amount of active time spent on developing the project. For example, there can be project which can be dormant for few years whereas another project even though new, can have active contribution leading it to be more mature. Even though this study evaluates maturity mainly based on the number of years that a project has been existent, it does not provide false positives as all the projects under study have been active all the years since their inception. But this study ignores the extent or frequency of activity of the projects as it is hard to measure.

	Emulab	*Ironic*	*Crowbar*	*Razor*	*Cobbler*	*MaaS*
Years Active	12	0/3	3	2	8	2

Table 1: Maturity of the project

From Table 1, we can infer that Emulab has been in use for the longest time and Cobbler has been the second longest survivor. Emulab is an academic project whereas Cobbler was first started as a part of RedHat Linux distribution's kick start server project, which has now become a standalone project. Crowbar has been in development and use for the last 3 years, whereas Razor and MaaS have been in use for 2 years, OpenStack Ironic is in use for less than a month, but its predecessor Nova Baremetal has been in use for around 3 years.

4.3 Number of deployments/users

Number of deployments and the number of users of a particular framework is a measure which can reveal useful insights like which framework is more popular among different organizations and the reason behind it. The number of deployments and users is hard to measure accurately. But there are other metrics like number of users in the mailing list, number of contributors, number of potential contributors or people of interest, number of big organizations which are using or backing the framework, etc. which can provide a useful approximation of the total number of deployments. It can also help in understanding the features or improvements that will be made over time.

	Ironic	*Crowbar*	*Razor*	*Cobbler*	*MaaS*
Users in mailing list	-	300	-	252	-
Contributors	77	53	13	130	25
Forks in Github	38	274	55	232	-

Table 2: Contribution and usage metrics

Table 2 gives the number of users in the developer mailing list, number of contributors to the project's Github repository and the number of times the project has been forked in Github. The '-' values in the table indicates that those metrics are unavailable for the particular project. For example, OpenStack does not give out the number of people subscribed to their mailing list.

Emulab has 38 major deployments around the world which mainly includes academia and research organizations [16]. Several clusters like Probe [17] and GENI [18] use Emulab as the underlying software. All the main contributors of Emulab are from Flux research group at Utah University and in the past year there have only been 9 contributors to the Emulab stable branch [19].

Dell Crowbar's main users and contributors include Dell, Rackspace, SUSE and some telecom and finance firms [5]. So far there are about 500+ downloads of the Crowbar software. From table 2, it can be seen that Crowbar has a comparatively lesser number of contributors but more number of people who have forked it from the Github repository. This tells us that the extensibility of Crowbar is comparatively higher that it allows people to develop their own modified versions. The number of people who have subscribed to the developer mailing list of Crowbar validates this further. Also the lesser number of contributors tells us that only the developers from inside Dell are the major contributors of the actual software. This is completely reversed in the case of OpenStack Ironic which has more contributors than there are forks, which speaks about a more diverse contribution environment.

Cobbler has stable number of contributors (130) and forks (232), which is in line with Crowbar. Also there are a healthy number of people who subscribed to Cobbler's mailing list. Some of the major users of Cobbler include Eucalyptus, Fedora and Ohio University [7].

Razor has a relatively lesser number of contributors (13) and forks (55), which directly reflects the number of years that Razor has been in business. But Razor is backed by EMC and Puppet Labs and has good enterprise level support.

OpenStack Ironic is used mainly by HP Cloud who are the primary contributors to the project alongside NTT Docomo. The number of contributors, as previously pointed out, can have both positive and negative effect. The high number of contributors indicate the openness of the project which can lead to a lot of people contributing in turn leading to rapid development. Also higher number of contributors who are from outside the organization means a lot of changes are made by people from outside the companies which are primarily responsible, which could lead to a much messier code base which can lead to bugs.

MaaS has a limited amount of contributors (25) who are mainly affiliated to Canonical, the company which produces Ubuntu. The number of forks could not be found because MaaS uses launchpad and not Github for version control.

4.4 Difficulty of installation

The difficulty of installation is another key component that needs to be considered in setting up a cluster using a baremetal provisioner. The following table shows the number of pages of installation document for each of the baremetal provisioner under study.

	Emulab	*Ironic*	*Crowbar*	*Razor*	*Cobbler*	*MaaS*
Number of pages	20	15	1	3	3	2

Table 3: Pages of installation document

Even though the number of pages in the installation documentation is not a valid metric to accurately measure the difficulty, it provides some assistance in this regard. For example, the installation documentation for Crowbar is only 1 page mainly because Crowbar has an inbuilt ISO image which when installed on a machine in a cluster will act as a baremetal provisioner. Then it can be configured to add multiple elements using the barclamps that are available for each of them. Razor is fairly simple to install because it can be installed as a puppet module which is automated. Further configuration can also be done by using puppet's modules which reduces the time involved in setup. MaaS is simpler mainly because of the ubuntu version that is offered with MaaS. That is, MaaS provides a completely configured ubuntu OS image which can be downloaded and booted to act as a MaaS baremetal provisioner with very little additional configuration. Cobbler is also fairly easier to install. Ironic is time consuming which can be inferred from the number of pages involved in the documentation. This is mainly because, for Ironic to be installed, it requires a set of other services like Nova (compute service), Glance (OS image storage), Keystone (authentication service) to be installed. Also some of these services require configuring databases, setting up network, etc. Emulab is probably the hardest of all the baremetal provisioners which can be inferred from the number of pages in documentation. This is because Emulab offers a set of features which are not available in other frameworks, and has stringent requirements on the operating system and packages needed to build and install Emulab, which makes it more difficult in setting up. A typical Emulab setup involves getting FreeBSD version 8.3 (the only supported OS type and version) and downloading a set of packages which needs to be installed on top of it [20]. Then the administrator has to download and build the code base and install it on different nodes. One setup for the master node and a different setup for the ops nodes. This is followed by a non-trivial database, web server and network configuration and setting up Emulab file server, image server, ops server and core server [21].

4.5 Stability

The stability of the project is directly dependent on the number of breaking changes or deprecations made to the project. Separate releases or incremental changes does not fall under breaking changes and deprecation of an entire version. OpenStack's baremetal has gone through 2 breaking changes in the past 3 years. The first was a move from what is now known as Nova Baremetal Historical to Nova Baremetal. The second move is the one currently in progress, from Nova Baremetal to Ironic. Emulab has not undergone major changes in the past 4 years. This is ascertained by looking at the commit logs of Emulab stable source branch. There are not enough data points available on the other provisioning frameworks to make a solid statement on their stability.

4.6 Difficulty of maintenance

All the frameworks under study have a web front end or UI from which we can control the operations of the provisioner. So management of the cluster through the provisioner is fairly simple in all the cases. But one of the problem with Ironic is that automatic discovery of nodes is not available at this point in time, but is in the pipeline for the future versions. That is, when a hardware is booted without an operating system and it has an Intel PXE enabled hardware, then it should be detected by the provisioner (bootstrap server) automatically without the user having to register the nodes to the provisioner. Crowbar, Razor, Cobbler, MaaS and Emulab automatically discovers the new hardware nodes. Razor and MaaS can launch an OS on the automatically discovered nodes if pre-configured with the required policies. This automatic discovery is not possible in Ironic at the moment and is planned for a future release. Without automatic discovery of new hardware, maintenance becomes tough as the administrator has to register all the machines to the provisioner (which in Ironic is called a Ironic Conductor). Another important maintenance issue is software upgrades. This directly depends on the stability of the project. If a project is not stable, then upgrades become much more difficult to perform as it will break a lot of things which was functioning properly prior to the upgrade.

4.7 Hardware requirements

Before installing any software we need to make sure that it works with the hardware that is available. In this case, a cluster of machines should have compatible hardware for the provisioning framework. All the baremetal provisioners require an IPMI or any other power manager enabled hardware which will allow the provisioner to use IPMI credentials to power up the machine. The newly started guest machine can then be bootstrapped using the default PXE boot mechanism as mentioned previously. Another usual hardware requirement in the case of baremetal provisioners is the network interface card (NIC) with VLAN capability which is needed in most cases to form a virtual network through VLAN once a set of nodes are allocated. Crowbar also needs a BMC (baseboard management control) which provides a more intelligent power management with a lot more functionality than a default IPMI.

4.8 Breadth of features

This section discusses some of the useful and important features that each of the framework has to offer over the others. The following features are chosen for analysis mainly because of their importance in a cluster or a testbed.

4.8.1 VLAN support

VLAN support is needed whenever we need to isolate a set of baremetal nodes into a separate virtual network. This support is available in all of the frameworks under study. Emulab has support for manually configuring VLAN as well as automating this process by specifying a network topology on node allocation. OpenStack Ironic supports VLAN setup on the baremetal nodes, but currently does not automate or provide assistance to do this. OpenStack's network component known as 'neutron' has support for this. But neutron is not completely integrated with Ironic. This requires VLAN to be configured manually. Razor, Cobbler, Crowbar and MaaS provide this support using their CMS solution. Razor has separate puppet modules which can be used to configure VLAN. Cobbler provides built in commands which enables easier and faster VLAN setup. Crowbar has a separate network barclamp which automates this task. MaaS allows users to handle VLAN configuration using its CMS juju.

4.8.2 Network Topology

The ability to setup network topology in a baremetal provisioning framework is very important, especially in a network testbed because, it can enable users to perform a lot of network level research on actual networks rather than only simulating them with a tool like NS2. This ability to create virtual networks with different topologies is lacking in all of the commercial baremetal provisioning frameworks [22], where we have to configure our own virtual network separately after we allocate a set of nodes. Emulab offers this feature which can be used to setup network topologies by specifying them in the NS file that is submitted when launching an experiment. The specification involves bandwidth requirements, connection and topology requirements, LAN or WLAN requirements, etc. Emulab also allows users to configure network topology using a GUI.

4.8.3 OS support

This section discusses the support for various operating systems in different baremetal provisioning frameworks. MaaS supports only Ubuntu operating system as it is released by the makers of Ubuntu and much of the functionality of MaaS is pre-configured into the Ubuntu OS image. Ironic and Cobbler has support for launching only Linux operating systems using the provisioner and no support for Windows. Crowbar, Emulab and Razor support both Linux operating systems and Windows.

4.8.4 High Availability

High availability is one of the main needs in any cluster or distributed system. High availability in a baremetal cluster means that even in the case of a node failure, the provisioner should be able to migrate the OS image to a different node and resume without any issues. This is easier in a virtual machine where taking a snapshot of the underlying memory is made easy by the hypervisor. But in baremetal machines only the snapshot of the disk can be taken. This is still valuable if a node can be reconstructed from its disk after failure. High availability is planned in MaaS in the latest edition of Ubuntu server version 14.04. High availability can be configured in Crowbar manually. Ironic does not have support for high availability right now but has it planned for future implementation. Emulab does not support automatic fail over needed for high availability, but has the option for preempting a node or a set of nodes [23]. This helps in relaunching the nodes with the exact disk state before which they were preempted. This process of taking snapshots is important in any time sharing and high availability system.

4.9 Performance

This section evaluates the performance of the baremetal frameworks Emulab and Ironic. The evaluation criteria for performance is provisioning different number of nodes and measuring the time taken from the start to completion of provisioning which is marked by powering on the operating system running on the machine.

4.9.1 Experimental Setup

Both the Emulab and Ironic experiments were run on a cluster with the server (provisioner) hardware configuration of 16 GB of memory and 1 Gb network connection. The provisioner ran an Ubuntu 12.04 LTS 64 bit operating system. The provisioned hardware was IPMI and PXE enabled.

4.9.2 Experiment

Fig. 2 shows the time taken to provision a single baremetal node using both Emulab and Ironic. It can be seen that the time to provision a node in Ironic takes slightly longer (26 seconds on an average) than Emulab. But we cannot conclude that Emulab performs better than Ironic as there are a lot of variables involved like the time taken for the request to redirect through the proxy which was present in the case of Ironic and not in Emulab, and the time taken to download the OS image over the network which depends on traffic at a particular point of time (as both the experiments were carried out on a different network even though the bandwidth of both the networks is 1Gb), time taken to boot the operating system, etc. The network transfer time of image is significant when compared to the time spent in other operations of the provisioning workflow. This is also the main reason for both the systems to have a fairly closer provisioning time. This can be further solidified by figure 3.

Fig 3 shows the time taken to provision different number of nodes in Emulab. It is in gradually increasing order and it takes 5 minutes and 50 seconds for provisioning 10 nodes. The very minimal increase in the provisioning time is due to the fact that all the nodes are provisioned in parallel. The increasing time is due to the fact that there can be differences in time taken to send the OS images over the network for each of the separate nodes and for the operating system to boot up on those machines before they are usable.

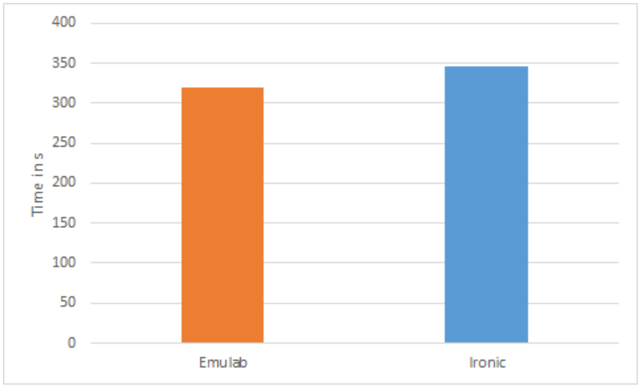

Figure 2: Time taken to provision 1 baremetal node

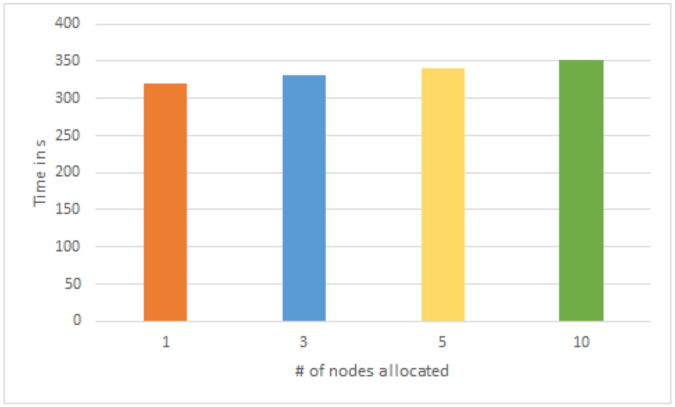

Figure 3: Time taken to provision baremetal nodes in Emulab

The performance measure as a whole is inconclusive. So the set of operations performed by both the systems to provision a baremetal machine is analyzed. Table 4 shows the steps involved in provisioning nodes in Emulab and Ironic. From the table we can see that the differences between both the workflows is that Emulab parses the NS file and stores information to the database on provisioning, whereas registering nodes happen before the provisioning begins in Ironic, the scheduling policy and the final network setup before deployment in Emulab. Once the deployment starts all the operations are similar for both Emulab and Ironic till the node is booted and running. Of these differences in the workflow mentioned above, time taken to parse the NS file and load it into a database should be minimal and there is no way to accurately measure it. The second difference is the network setup step in Emulab. Most of the operations in this step is configured as a delayed event. A delayed event is one which takes place at a later time asynchronously and causes the main workflow to return immediately [19]. This also has minimal impact and is not accurately measurable, which leaves us with scheduling policy which can have a considerable impact on performance, correctness and future schedules. The next section evaluates the scheduling policy adopted by both the systems.

4.10 Scheduling

As discussed earlier, scheduling is a NP hard problem and is important in deciding the resource utilization of the cluster and performance of the provisioner. Scheduling mainly becomes an issue when there are multiple baremetal nodes that needs to be allocated from the list of available nodes, because the possibility of a schedule wasting resources is pretty high due to fragmentation of resources. The NP hard nature of scheduling suggests that there is no possible optimal solution to the problem. As a result each of the baremetal provisioning frameworks has its own scheduling algorithm based on its priorities. Any of the uncited claims about the working and implementation of Emlulab in the subsequent sections can be attributed to the author's understanding based on the source code of Emulab [19].

Emulab	Ironic
Parse the NS file and store information on to database	Get the set of registered nodes from the database
Load the information about physical nodes and requested nodes from the database	Run the filter scheduler to choose the specific nodes for scheduling
Run the simulated annealing scheduler to choose the specific nodes for scheduling	Get the image from Glance database and initiate deployment
Convert the link, switch and node information into actual VLAN and port group with some operations delayed to be completed later	
Get the image file and initiate deployment	

Table 4: Steps involved in provisioning

Emulab like mentioned before creates a virtual network among the allocated nodes based on the network topology specified by the user. This increases the number of variables involved in making a scheduling decision [1]. Some of the variables involved include bandwidth of the network connection between two nodes which varies based on the topology that the user selects as the actual physical bandwidth has to be shared, special requests made as a part of allocation (like node with GPU, etc), using a special node for a non-special case (allocate a node with GPU for an experiment which does not require GPU), weights assigned to different requirements, etc. To solve this problem of creating an optimal network virtualization on top of the physical network, Emulab uses a simulated annealing approach [24]. Simulated annealing is a randomized search algorithm which takes a cost function and a generator function, and at each level a generator function changes the given set of parameters into a new set of parameters which are evaluated against the cost function to see if the cost of the new set of parameters are lower than the previous step, and proceeds based on that. The algorithm keeps converging and terminates on reaching an objective function [25]. In Emulab's case, the simulated annealing algorithm is a graph mapping problem. Emulab takes a virtual graph (network topology with specific needs for bandwidth and connection between nodes) and tries to place it on top of the physical graph (actual network topology of the nodes in the cluster). The cost function is based on weights assigned to the variables in scheduling like bandwidth of links, special requirements, etc. The generator function or the transition function is the neighbor selection in the physical graph which increases the search space at each step. The objective function or the verifying function consists of two constraints. First one is the validity constraint which checks whether the schedule is valid (meets the basic requirements). Second one is the optimality constraint which verifies whether the cost is within a certain bound. Emulab further optimizes the algorithm by applying the concept of equivalence classes to reduce the search space considerably and by flagging unacceptable schedules by checking against a configuration file even before starting simulated annealing. This Emulab scheduling algorithm is fixed and cannot be modified or configured by the user or the administrator. This loss in configurability may not be suitable for some users. On the other hand, the optimal scheduling algorithm is more than sufficient for most users.

Crowbar also uses simulated annealing algorithm for baremetal node scheduling [26], which is similar to the algorithm that Emulab uses. But Crowbar's needs are different compared to Emulab as it does not have a network topology to lay on top of the physical cluster. Instead it characterizes its scheduling algorithm as a minimization of resources problem. Emulab uses C++ which is faster compared to the ruby library that Crowbar uses.

Ironic uses nova's filter scheduler for managing baremetal nodes. OpenStack has a controller node which acts like a master and a set of compute nodes which act like cluster controllers which keep track of the resources available. Ironic has a component known as Conductor which runs at the compute node and aggregates the available resources and sends it to the nova scheduler which runs in the controller node. After a new baremetal node is registered with its MAC address, it can be launched using Ironic. On launching, the scheduler makes a decision based on its view of the number of available nodes and its resources and the filters that are already set. Filters are constraints which has to be satisfied for a node to get allocated. There are some default filters provided by OpenStack and custom filters which can be created by the user and configured. The scheduler computes a weighted sum for each of the available nodes based on the weights assigned to each of the filter values and the resources and chooses a schedule with the highest weighted sum. The scheduler then contacts the appropriate conductor running on a compute node to schedule the node. Ironic also provides the option of changing the default filter scheduler to a chance scheduler which randomly picks the eligible hosts and does not compute a weighted average like filter scheduler does.

4.10.1 Experimental Setup

The three scheduling policies discussed above viz. simulated annealing scheduler, filter scheduler, chance scheduler were evaluated for performance. All these three schedulers were run against a test data set which was generated based on a ruby script. The ruby script was implemented such that it randomizes the physical nodes and their configuration. In the case of filter scheduler the script also generates a set of random weights and filter options. In the case of simulated annealing scheduler for Emulab, the script also generates a set of network configuration for the nodes which include bandwidth, connections, etc. The script is also capable of generating the configuration of the nodes that need to be allocated. The script gets the number of nodes in the cluster and number of nodes to be allocated as parameters and generates them based on the scheduler which will use that input. In the experiment conducted a 1000 node cluster was simulated using the script. Emulab's simulated annealing scheduler known as assign was ran separately with the simulated cluster configuration generated by the script, and the time taken to completion and # of iterations run were measured. Nova's filter scheduler was run using the same configuration and a set of randomized filters. A chance scheduler was implemented which randomizes the selection once the set of nodes match the required criteria. It was also tested against the same configuration generated by the script. The hardware used for the experiment is a 1.7 GHz single CPU with 2 GB of RAM. The operating system used was a FreeBSD 8.3 32 bit version. It was chosen as it was the only OS which will run Emulab's scheduler. All the randomizations mentioned above (machine configuration, filters, etc.) were performed using ruby's sample method which samples a finite set of possible values. The running time of all the three schedulers mentioned above were measured which is summarized in figure 4.

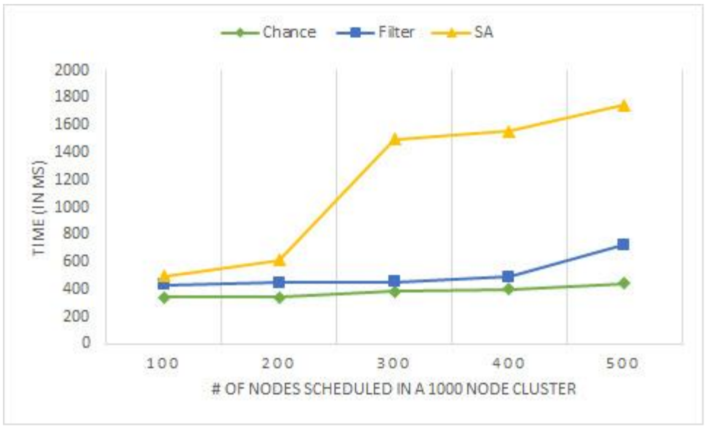

Figure 4: Time taken to schedule nodes

From fig. 4, we can see that the time taken to schedule different set of nodes in a 1000 node cluster for a filter scheduler is slightly higher than that of a chance scheduler (100 ms approx.). There is only minimal difference because the randomize function is costly and a filter scheduler's weighted average does not cause much difference in the running time for a smaller set of nodes (1000). In the case where 500 nodes are scheduled out of the 1000 available in the cluster, the running time is markedly higher for filter scheduler (725 ms). This is due to the fact that the filter scheduler was not able to find a viable schedule after running to completion in that particular case. We can also infer that simulated annealing is costlier than filter scheduler and chance scheduler as expected and it steadily rises with the increase in the number of nodes to be scheduled. It takes 1.7s to schedule 500 nodes out of 1000 nodes in a cluster. This is still fast for a machine learning algorithm. This can be due to several reasons. The language of implementation is C++ which is faster compared to the other schedulers which were implemented in python and ruby. Emulab's assign also performs optimizations like equivalence classes which groups nodes into groups and filters them together instead of checking each node. Another main reason behind this is that the node configuration and network topology generated is normal as it is randomly generated. If we specifically assign a lot more special requirements in terms of bandwidth, links, weights, etc. then the algorithm will take considerably longer to complete.

Each of these schedulers differ in performance, usage and the extent to which they can be configured. Based on these parameters and the analysis above the administrator can choose a baremetal provisioning framework which will be ideal for the situation.

Criteria	*Emulab*	*Ironic*	*Crowbar*	*Razor*	*Cobbler*	*Maas*
Vendor Lock-In	O	O	O	O	O	O
Maturity	O	X	-	-	-	-
User Base	O	X	O	-	-	-
Active Contribution	X	-	O	-	-	-
Stability	O	X	-	-	-	-
Difficulty Of Installation	X	X	O	-	-	O
Difficulty Of Maintenance	-	X	-	-	-	-
Configurability Of Scheduler	X	O	-	-	-	-
VLAN Support	O	O	O	O	O	O
Network Topology	O	-	-	-	-	-
OS Support	O	-	O	O	-	X
Automatic Discovery	O	X	O	O	O	O
Hardware Requirements	-	-	X	-	-	-
High Availability	O	X	-	-	-	-

Table 5: Overall metrics

Table 5 shows the cumulative report of all the categories that are used to evaluate the different baremetal frameworks. 'O' indicates best (based on the quantitative measure provided in the corresponding section) in the category, 'X' indicates worst (based on the quantitative measure provided in the corresponding section) in the category among the six baremetal provisioning frameworks and '-' indicates neither best nor worst. This table is a summary based on the discussion from the evaluation section.

5 Conclusion

This section will summarize each of the baremetal provisioning frameworks on its strong and weak points based on the data points collected and my experience with the system.

5.1 Emulab

Emulab is more suitable when we need a network or distributed systems testbed, as it offers virtual networks and a lot of other network level customizations which currently no commercial vendor offers [22]. Emulab is open source and free to download and install. It also provides a way to snapshot, preempt and restart, which is a great feature to have especially in a research testbed where experiments are run. On the downside, if we need a simple cluster manager for a cluster which will mainly involve installing an operating system and other tools, then Emulab might not be the right fit due to the difficulty in installation which can be very easy in a system like MaaS, Crowbar or Razor. Another disadvantage of Emulab is the non-availability of a configuration management system like Chef, Puppet, etc.

5.2 Ironic

Ironic does not offer testbed like features of Emulab. Also at this point in time it lacks stability and some common features which are present in other baremetal provisioners like automatic discovery and high availability. Another inference from manually setting up Ironic is that, the documentation is not comprehensive and a lot of questions in mailing list are about bugs in code which breaks the setup, failing unit tests, etc. This can be attributed to OpenStack's style of software releases. That is, any OpenStack project will be released in a short time span and will be made better incrementally. Also some of this can be attributed to the number of non-professional contributors who contribute to the code base [Section 4.3].

But Ironic is important mainly because it can make OpenStack self-sufficient. That is, a hardware cluster can be run only using OpenStack without depending on any other software. This is possible by launching baremetal nodes using Ironic and setting up OpenStack cloud on top of those machines. Currently this auto deployment of OpenStack is being done by other tools like Crowbar and Razor. Ironic also has plans for supporting some of the common features like high availability and automatic discovery in the future. Automatic discovery is proposed to be implemented using a consistent hashing approach where each node forms a part of the hash ring and whenever a node gets added to the cluster or gets unregistered, it can be automatically updated [4].

5.3 Other frameworks

Crowbar is widely used to launch OpenStack cloud using the predefined barclamps. It is really stable and more comfortable to use than Ironic or other OpenStack deployment software. But it has some specific hardware requirements like BMC which are present in Dell machines. Cobbler has been in development for a long time and is stable enough for use in any big cluster deployments. Razor and MaaS are fairly recent but solve some of the problems with existing systems like a fully functional web UI and easy deployment respectively.

6 References

[1] Richard Potter, Haider Aun and Akihiro Nakao. Challenges in resource allocation in network virtualization. *20th ITC Specialist Seminar*, 18(20), 2009.

[2] Charlie Wiseman and Jonathan Turner. The virtual network scheduling problem for heterogeneous network emulation testbeds. *Washington University in Saint Louis, Tech. Rep*. WUCSE-2009-68, 2009.

[3] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Operating Systems Review*, 36(SI):255-270, December 2002.

[4] Openstack ironic. https://wiki.openstack.org/wiki/Ironic.

[5] Dell crowbar. http://crowbar.github.io/home.html.

[6] Canonical metal-as-a-service. http://maas.ubuntu.com/.

[7] Cobbler. http://www.cobblerd.org/.

[8] Puppet lab's razor. http://puppetlabs.com/solutions/next-generation-provisioning.

[9] Apache hadoop. https://hadoop.apache.org/.

[10] Opscode chef. https://wiki.opscode.com/display/chef/Home.

[11] Puppet labs puppet. http://puppetlabs.com/.

[12] Canonical juju. https://juju.ubuntu.com/.

[13] Openstack icehouse release. http://www.openstack.org/software/icehouse/.

[14] Crowbar project home. http://crowbar.github.io/home.html.

[15] The University of Alabama Case Study. http://www.dell.com/learn/us/en/uscorp1/customer-stories?c=usl=ens=corp.

[16] The UX research group in the University of Utah. http://www.ux.utah.edu/.

[17] Probe testbed. http://www.nmc-probe.org/.

[18] Geni testbed. http://www.geni.net/.

[19] Emulab stable release branch. http://git-public.ux.utah.edu/gitweb.cgi?p=emulab-stable.git;a=tree.

[20] Emulab setup. https://wiki.emulab.net/wiki/install/required$_s$oftware:html:

[21] Emulab complete installation guide. https://wiki.emulab.net/wiki/install.

[22] Jiayue He, Rui Zhang-Shen, Ying Li, Cheng-Yen Lee, Jennifer Rexford and Mung Chiang. Davinci: Dynamically adaptive virtual networks for a customized internet. *In Proceedings of the 2008 ACM CoNEXT Conference*, *CoNEXT '08*, pages 15:1-15:12, New York, NY, USA, 2008. ACM.

[23] Mike Hibler, Robert Ricci, Leigh Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. *In USENIX Annual Technical Conference*, pages 113-128, 2008.

[24] Robert Ricci, Chris Alfeld and Jay Lepreau. A solver for the network testbed mapping problem. *ACM SIGCOMM Computer Communication Review*, 33(2):65-81, 2003.

[25] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. *Springer*, 1987.

[26] Simulated annealing based scheduling in open crowbar. https://github.com/opencrowbar/core.