International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-8 Issue-2, July 2019

Scheduling under Open stack – The Current State and Future Enhancements

Uma Maheswara Rao I, JKR Sastry

Abstract*: Cloud computing is being heavily used for implementing different kinds of applications. Many of the client applications are being migrated to cloud for the reasons of cost and elasticity. Cloud computing is generally implemented on distributing computing wherein the Physical servers are heavily distributed considering both hardware and software, the connectivity among which is established through Internet. The cloud computing systems as such have many physical servers which contain many resources. The resources can be made to be shared among many users who are the tenants to the cloud computing system. The resources can be virtualized so as to provide shared resources to the clients. Scheduling is one of the most important task of a cloud computing system which is concerned with task scheduling, resource scheduling and scheduling Virtual Machin Migration. It is important to understand the issue of scheduling within a cloud computing system more in-depth so that any improvements with reference to scheduling can be investigated and implemented. For carrying in depth research, an OPEN source based cloud computing system is needed. OPEN STACK is one such OPEN source based cloud computing system that can be considered for experimenting the research findings that are related to cloud computing system. In this paper an overview on the way the Scheduling aspect per say has been implemented within OPEN STACK cloud computing system.

***Index Terms*: Cloud Computing, Scheduling, OPEN STACK, Resource scheduling, Task scheduling, VM migration scheduling**

I. INTRODUCTION

Scheduling is one of the most important tasks that have many dimensions to be fulfilled when it comes to Cloud computing System. Generally processing tasks are to be scheduled as all of the tasks cannot be executed in single go. A single CPU can execute instructions related to a single program in subsequence considering program flow. Task swapping is done considering time sharing, priority assigned to the tasks, voluntary relinquishing the processor such that all the tasks will have the accessibility of the processor so that task related instructions can be executed by the processor. While there is a single processor, there could be many tasks that must be executed, in which case scheduling of the tasks must be done based on some criteria which include size of the task, least CPU time consumed by the task, priority of the tasks, based on the sequence in which the tasks are initiated for execution etc. task switching is the most important criteria without which all the tasks that are to run will not get the opportunity to be run. In multi-programming, multi-tasking, the need for running several tasks arises which is made possible by keeping several tasks in one of the processing states. These days, hardware servers with multi core architecture have come up heavily which means many processors could exist in a physical server. Scheduling of the tasks to run under multi-processor architecture is complicated as the tasks are to be scheduled considering several processors situated in a Physical server. In distributed computing, physical severs exists as a cluster, each server may be of single processor or multi-processor architecture. Scheduling of the tasks becomes much more complicated when number of physical servers are more and each physical sever is made of Multi-server architecture. Scheduling under cloud computing is much more complex. In cloud computing scheduling is related many aspects that include scheduling tasks, Scheduling resources, scheduling Migration, Scheduling provisioning etc. The concept of Virtualization is implemented within Cloud computing so as to implement different service models that include IaaS, PaaS, SaaS. Different kinds of services are made available to the end user through the concept of Virtual Machines to which Virtual resources attached and the users are provided with the desired service through Virtual machines. So the issues of scheduling under clouds have many dimensions and in each dimensions several approaches to deal with the issues of scheduling exists. An overview of the scope of scheduling within cloud computing system becomes quite important. OPEN STACK is a cloud computing system which is an OPEN source made available for the researchers to implement their own cloud computing based applications and various research findings. It is important to understand the way the issue of Scheduling is handled within open stack so that new investigations and findings related to scheduling can be experimented by making changes to the existing architecture of the OPEN STACK cloud computing system. Users can interact with OPEN Stack system using HORIZON components which provides dash board which provides for interface communication, computing, selection of images and for providing storage related services. The Interface related sub-components existing in the Dash board is is shown in Figure 1.

Revised Manuscript Received on 30 July 2019.
***** Correspondence Author
Dr. JKR Sastry*, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur District, AP, India
Uma Maheswara Rao I, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur District, AP, India.

The Block diagram showing the components existing the compute node, the middleware used for effecting between the components and the hardware is shown in Figure 2.

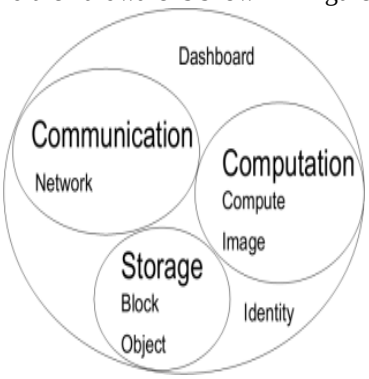

Figure 1 OpenStack Dash Board

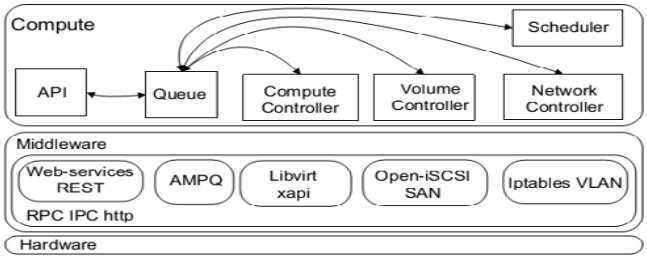

Figure 2 Compute Service and the Underlying Platform

OpenStack is composed of seven main services which can be grouped into three principal areas: Communication, Storage, and Computation as depicted. These services are backed by two support services, namely Identity and Dashboard. Compute service manages the virtual disks and associated metadata in Image. Dashboard provides a web-based front-end to the Compute whereas Network provides virtual networking for Compute. Block Storage provides storage volumes for Compute. Image can store the actual virtual disk files in the Object Storage and all the services authenticate with Identity. The Compute service consists of Web-based API that ensures the command and control aspects of the computation, storage and networking. The component Queue brokers the interaction between the Compute service components, i.e., Volume, Network and Compute Controllers, Scheduler and API. Compute Controller manages the life-cycle of computing instances (i.e. VMs) on the nodes. The Network Controller manages the networking resources and the Volume Controller ensures the interaction between the instances and the Block Storage. The sequence diagram that shows the way interaction between the components over their life cycle is shown in Figure 3.

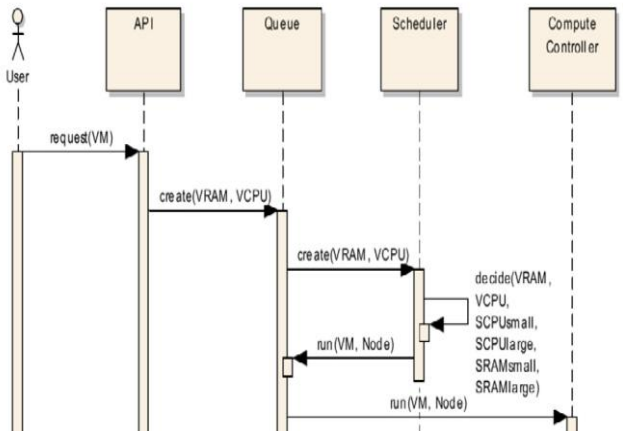

Figure 3 Sequence Diagram – Interaction between compute components

The underlying platform is composed of the middleware and the hardware. The middleware includes several components such as Web-services, AMPQ, etc. These components are communicating using normalized protocols, such as RPC, IPC and http, and sharing the same hardware layer. Once API receives user issued request for launching an instance (VM), it interprets this request and send it to the Queue that is providing messaging between all other components of Nova. After receiving the request for creating an instance, Scheduler decides which Node will get the instance according to the available resources (the amount of free memory and the number of available CPU on nodes) and the specification in the request (the memory and the number of CPU). The Queue dispatches this association between an instance and the Node to the Compute Controller which is launching the requested VM on one particular Node. OpenStack Compute includes three types of Scheduler: Filter, Chance and Simple. A filter is a two-stage process that supports filtering and weighting using predefined costs and their associated weights in order to schedule the deployment of instances (i.e. VM) on some physical node. The Chance Scheduler chooses randomly an available node regardless of its characteristics while the Simple scheduler tries to find an available node with the least load. There is a need to understand the issue of scheduling, the way it is implemented within OPEN STACK so that researchers better know how the OPEN stack can be improved through implementation of new methods, algorithms and architectures. Clear understanding of the OPEN stack will help the organizations to implement their own private or public clouds.

II. RELATED WORK

Oleg Litvinski et al., [1] have developed an experimental test bed using which the working of the OPEN stack scheduler can be assessed. They have designed an experiment with the main aim of gathering information about the functioning of the scheduler. The experiment is deigned to study the effect of screening factors such as number of CPUs, amount of memory etc. on the performance of the scheduler. Enforcing the Real time processing within a Cloud computing system is challenge. An attempt has been made by Sisu Xi et al., [2] for transforming OPEN stack as Real time open stack which is called as RT-OPEN Stack. They have introduced into the OPEN stack architecture three basic components that turn OPEN stack to real time OPEN Stack which include a real-time Hypervisor, a cloud management system through real time interface, a real time scheduler to allow regular VMs to share HOSTS, a VM to HOST Mapping strategy. Lianhao Lu et al., [3] have presented the issue of scheduling the user requests considering various performance issues which is dependent on the workloads. The have presented that the resources can be better utilised by proper scheduling and resource over committing. OPEN stack allows the administrators to configure resource scheduling based on the usage of the resources. The authors in their paper, have shown how maximum resource utilisation can be achieved through making changes to the scheduling process. U.Vignesh et al., [4] have shown how changes that must be made to NOVA scheduler such that enterprise applications can be implemented with the OPEN stack. The Nova scheduler in Open Stack is customized according to the requirement of the organization which develops the cloud. The cloud is capable of providing the infrastructure as a service which includes operating systems and middleware environment.

Feng-Jun Shang [5] has presented An Initial Resource Scheduling Algorithm for OpenStack.

Many intricate problems exist for scheduling various resources contained in the cloud computing system. The have presented initial resource scheduling algorithm based on matching hungry degree which is the degree of matching between the VM related resource requirement Vs availability of the resource on the physical machines. They have also considered the hungry matching degree of all the Virtual machines considering all the physical machines. The algorithm proposed by them finds all matching VMs with physical servers so that the VMs are optimally placed. They have shown the working of the algorithm in simulation basis but not directly implementing the same on OPEN stack. Haibo Li1 et al., [6] have presented that the in a typical cloud computing system, the queue sizes are fixed, and the use of the resource is static. They have shown how the batch processing systems can be used in conjunction with Cloud computing systems. They have presented the process of dynamic scheduling of the Virtual machines within open stack as per the Job queue. They have proposed a virtual cluster system that combines the Batch Jobs and the cloud based JOBS. Mohan Krishna Varma et al., [7] have presented a new technique of weighing the servers / processors along with number of tasks that are contained in the Job queue as a part of OPEN stack scheduler. They have presented that the OPEN stack implements the Filter scheduling algorithm for scheduling the virtual machine to the HOST. The Filter scheduler selects the hosts with highest weight and assigns the VMs to the highest weighted HOSTS. The algorithm does not consider the weight of virtual machine weight. The authors have proposed an algorithm that considers both the weights of VMs and Physical HOSTS and implemented the same within filter scheduling, round robin scheduling and Greedy scheduling. Sanhita Sarkar et al., [8] have presented techniques for optimising the scheduling of virtual machine. They have presented that to reduce energy consumption and carbon emission of the computing resources, the utilization rate of computing resources must be enhanced. They have opined that the distribution of physical resources and proper dispatching of the Virtual machine is one most important consideration of the cloud computing system. They have proposed multi-objective ant colony optimization algorithm for multi-objective optimization and redesigned the resource scheduling considering user specific requirements. They have implemented their finding on OPEN stack. They have compared their algorithm with other algorithm that include first fit algorithm, least residue algorithm and conventional ant colony algorithm and shown the performance improvements achieved in scheduling the cloud based resources. The algorithms have been analyzed considering violation rate of service level agreement (SLA), resource balance and placement superiority. They have found that first fit algorithm was moderate in service quality and energy consumption, but the randomness was high; least residue algorithm was excellent in service quality, but the energy consumption was high; the conventional ant colony algorithm and the improved ant colony algorithm had excellent service quality and low energy consumption. The analysis of the placement superiority of the four algorithms suggested that multi-objective ant colony optimization algorithm had good balance in service quality and energy consumption. The algorithm proposed by them considered multi-objective ant colony optimization which is proved to be efficient and that can be used to reduce energy consumption on the premise of ensuring service quality. Excellent placement of the computing resources is the most important issue that must be taken into consideration when it comes to organizing a proper cloud computing system. In the case of OPEN STACK cloud computing system scheduler takes into account multiple constraints that include processing and memory requirements. The native scheduling algorithm does not consider the network requirements of the Virtual machines nor the networking resources that are actually available within the cloud computing infrastructure. Michael Scharf et al., [9] have presented an extension of OPEN STACK scheduler that considers in addition to the processing and storage requirements, the networking requirements of VMs and the actual networking resources available. They have proposed network-aware placement of instances that takes into account bandwidth constraints to and fro from the nodes. The solution proposed by them keeps track of host-local network resource allocation, and it can be combined with bandwidth enforcement mechanisms such as rate limiting. They have also presented a prototype that requires only very few changes in the OpenStack open source software. Testbed measurement results demonstrate the benefit of their solution compared to the OpenStack default approach. Several types of scheduling algorithms have been proposed in the literature that look at scheduling tasks, VM migrations, load balancing, bandwidth utilization.[10][11][12][13][14]][15][16][17][18][19][20][21][22][23][24][25][26][27][28] but of them as such have not been implemented on real cloud computing system. Rather the efficiency of the proofs is based on simulation studies

III. INVESTIGATIONS AND FINDINGS

3.1 *Overview On Open Stack Components*

OPEN STACK has been built using many of the components some which directly provided the services and resources that are required by the end customer. Open stack consists of several services dedicated to specific functionalities. Swift can be defined as a file system. This service stores large amounts of unstructured data via a restful http api. Cinder provides persistent storage for virtual machines. When a virtual machine is destroyed, this storage is maintained. Glance is the registration and delivery service of virtual images for launching virtual machines. Finally, keystone is the service that manages identity and authorization management (iam). Keystone centralizes authentication and credentials of users and services. It guarantees the validity of this information by providing tokens that are recognized by the other open stack services. Keystone does not manage authorization policies of other services. Each service, includes keystone, decides whether to grant access or not to its resources according to its own policy and the received tokens validated by keystone. Nova is designed to create and manage virtual machines. It is compatible with the majority of virtualization technologies (xen, kvm, hyper-v, and lxc). Interaction with nova is performed through an api or a dashboard (component horizon).

3.2 Scheduling under OPEN Stack:

Nova –Scheduler is the component responsible to decide which NOVA compute node should be used when a VM instance is to be lunched. Nova Scheduler interacts with other components through queue. Queue is the essential data structure used for implementing scheduling of the tasks. Nova Scheduler collects information related to various resources and maintains them in a file called nova.conf. Nova Implements different kind of schedulers that suits to different filtering situations as per the need. Filter Scheduler is the most frequently used filter.

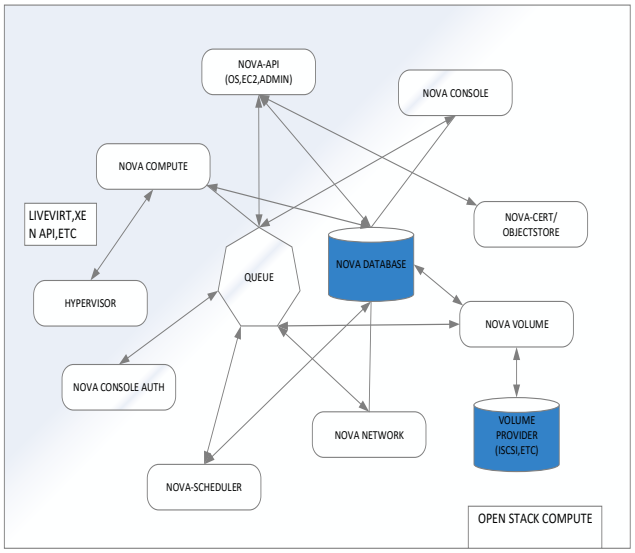

Figure 4 Compute Node resident component Interaction Diagram

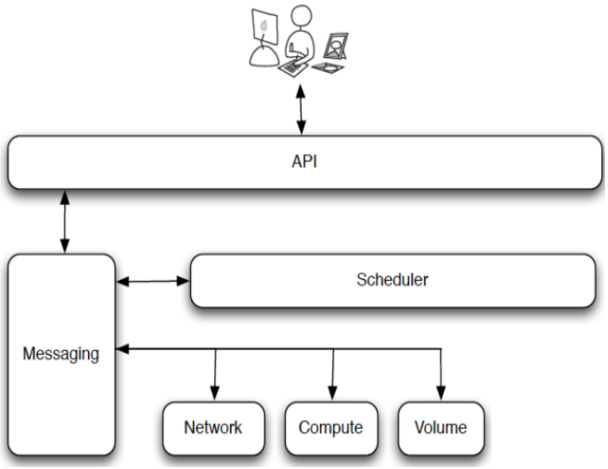

Figure 5 OPEN STACK - NOVA Architecture

NOVA Scheduling Process

The NOVA Scheduling process is divided into 3 Phases that Include (1) Getting the current state of all the compute nodes: it will generate a list of hosts, (2) Filtering phase will generate a list of suitable hosts by applying filters, (3) Weighting phase will sort the hosts according to their weighted cost scores, which are given by applying some cost functions. The sorted list of hosts is candidate to fulfil the user requests. The Nova Scheduling process is shown in Figuer-6.

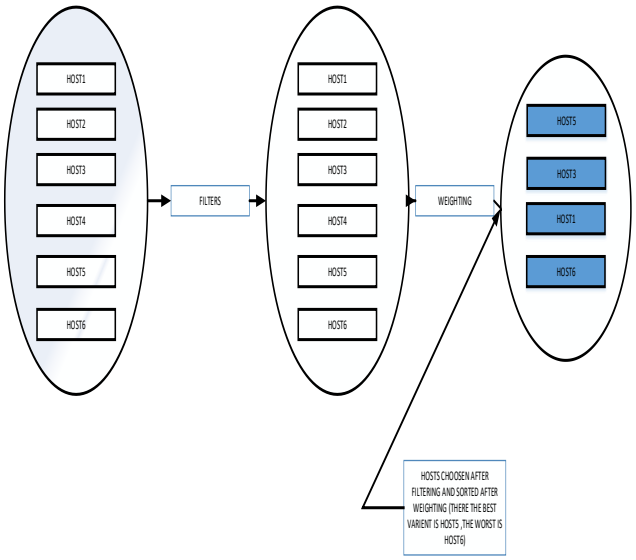

Figure 6 the scheduling process within OPEN Stack

NOVA processes the User requests sequentially on the FIFO basis. All the user requests are queued and the requests are processed as they come in. No priorities are considered by the scheduler for stacking the requests in the queue. NOVA scheduler will queue the user requests only when the user requested resources are available. A response is sent to the user saying that the requested resources are not available. The details of all the available resources are stored in NOVA-database. In open stack the resources are arranged in partitions and the user is assigned with the desired resources as a partition and no resource is individually assigned. Individually. Re-scheduling or re-allocation of the resources after completion of the usage of the resources needs to be done by administrator-component. NOVA scheduler is concerned with queuing the user request and uses and processing the same in the sequencing in which the requests have been received. A sample list of resource Partitions and the status of those partitions are stored in NOVA database. Table -1 shows the sample list of partitions.

3.3 Support of Filters by NOVA

Open Stack supports various kinds of scheduler filters. A specific filter can be chosen by the implementer through customization of nova.config file. Each of the filter is designed to meet certain amount scope related scheduling the resources. Some of the filters include AllHostsFilter, mageProperties Filter, AvailabilityZoneFilter, ComputeCapabilitiesFilter etc.

3.4 NOVA Scheduler Algorithms / Steps

The Filter Scheduler is the default scheduler and is used most frequently. Filter Scheduler supports filtering and weighting to make informed decisions on where a new instance should be created and the kind of partitions that must be attached to the virtual machines. Filter Scheduler iterates over all found compute nodes, evaluating each against a set of filters. The list of resulting hosts is ordered by weights. The Scheduler then chooses hosts for the requested number of instances, choosing the most weighted hosts. For a specific filter to succeed for a specific host, the filter matches the user request against the state of the host plus some extra magic as defined by each filter

If the Scheduler cannot find candidates for the next instance, then the user is informed of the lakh of hosts to run the VM instance. The Filter Scheduler has to be quite flexible so that it can support variety of filtering and weighting strategies. The user can also develop his/her own filtering strategy and make the system follow the strategy. However many filters have been already found and implemented that suits different filtering requirements. The way the scheduling is done is shown in Figure 7.

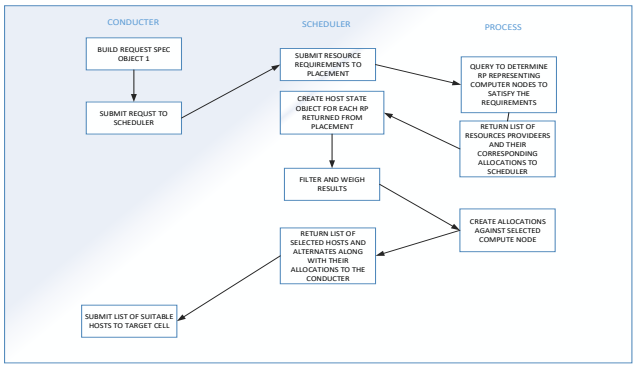

Figure 7 Scheduling sequences of steps

Following are the sequence of steps of processing undertaken for scheduling the resources

1. Scheduler gets a request specification from the "super conductor", containing resource requirements. The "super conductor" operates at the top level of a deployment, as contrasted with the "cell conductor", which operates within a particular cell.
2. Scheduler sends those requirements to placement.
3. Placement runs a query to determine the resource providers (in this case, compute nodes) that can satisfy those requirements.
4. Placement then constructs a data structure for each compute node as documented in the specification. The data structure contains summaries of the matching resource provider information for each compute node, along with the AllocationRequest that will be used to claim the requested resources if that compute node is selected.
5. Placement returns this data structure to the Scheduler.
6. The Scheduler creates HostState objects for each compute node contained in the provider summaries. These HostState objects contain the information about the host that will be used for subsequent filtering and weighing.
7. Since the request spec can specify one or more instances to be scheduled. The Scheduler repeats the next several steps for each requested instance.
8. Scheduler runs these HostState objects through the filters and weighers to further refine and rank the hosts to match the request.
9. Scheduler then selects the HostState at the top of the ranked list, and determines it's matching AllocationRequest from the data returned by Placement. It uses that AllocationRequest as the body of the request sent to Placement to claim the resources.
10. If the claim is not successful, that indicates that another process has consumed those resources, and the host is no longer able to satisfy the request. In that event, the Scheduler moves on to the next host in the list, repeating the process until it is able to successfully claim the resources.
11. Once the Scheduler has found a host for which a successful claim has been made, it needs to select a number of "alternate" hosts. These are hosts from the ranked list that are in the same cell as the selected host, which can be used by the cell conductor in the event that the build on the selected host fails for some reason. The number of alternates is determined by the configuration option scheduler.max_attempts.
12. Scheduler creates two list structures for each requested instance: one for the hosts (selected + alternates), and the other for their matching AllocationRequests.
13. To create the alternates, Scheduler determines the cell of the selected host. It then iterates through the ranked list of HostState objects to find a number of additional hosts in that same cell. It adds those hosts to the host list, and their AllocationRequest to the allocation list.
14. Once those lists are created, the Scheduler has completed what it needs to do for a requested instance.
15. Scheduler repeats this process for any additional requested instances. When all instances have been scheduled, it creates a 2-tuple to return to the super conductor, with the first element of the tuple being a list of lists of hosts, and the second being a list of lists of the AllocationRequests.
16. Scheduler returns that 2-tuple to the super conductor.
17. For each requested instance, the super conductor determines the cell of the selected host. It then sends a 2-tuple of ([hosts], [AllocationRequests]) for that instance to the target cell conductor.
18. Target cell conductor tries to build the instance on the selected host. If it fails, it uses the AllocationRequest data for that host to unclaim the resources for the selected host. It then iterates through the list of alternates by first attempting to claim the resources, and if successful, building the instance on that host. Only when all alternates fail does the build request fail.

The scheduler has become tightly coupled with the rest of nova, limiting its capabilities, accuracy, flexibility and maintainability. The goal of scheduler evolution is to bring about a better separation of concerns between scheduling functionality and the rest of nova.

3.5 Configuring Filters

Filters can be set by the users by setting the parameter filter_scheduler.available_filters assigned with the specific filter that must be used. The setting is done in the parameter file. All the filters that are configured can be used by OPEN stack during run time. With these settings, nova will use the Filter Scheduler that has been configured.

3.6 Designing Weights

Filter Scheduler uses the so-called weights during its work. A weigher is a way to select the best suitable host from a group of valid hosts by giving weights to all the hosts in the list.
In order to prioritize one weigher against another, all the weighers have to define a multiplier that will be applied before computing the weight for a node.

All the weights are normalized beforehand so that the multiplier can be applied easily. Therefore the final weight for the object will be:

weight = w1_multiplier * norm(w1) + w2_multiplier * norm(w2) +

A weigher should be a subclass of "weights. Base Host Wegher" and they can implement both the "weight multiplier" and" weight_ object" methods or just implement the "weight objects" method. "weight objects" method is overridden only if you need access to all objects in order to calculate weights, and it just return a list of weights, and not modify the weight of the object directly, since final weights are normalized and computed by weight. Base Weight Handler.

3.7 Weighers supported within NOVA

The Filter Scheduler weighs hosts based on the config option "filter_scheduler.weight_classes", which defaults to " nova.scheduler.weights.all_weighers". Nova supports many other weighs that include RAMWeigher, CPUWeigher, DiskWeigher etc.

Filter Scheduler makes a local list of acceptable hosts by repeated filtering and weighing. Each time it chooses a host, it virtually consumes resources on it, so subsequent selections can adjust accordingly. It is useful if the customer asks for a large block of instances, because weight is computed for each instance requested.

3.7 Limitations of the current Nova Scheduler

Many users want to do more advanced things with the scheduler, but the current architecture is not ready to support those use cases in a maintainable way. The following issues cannot be addressed with the current form of NOVA scheduler.

1. Cross Project Affinity

It can be desirable, when booting from a volume, to use a compute node that is close to the shared storage where that volume is. Similarly, for the sake of performance, it can be desirable to use a compute node that is in a particular location in relation to a pre-created port.

2. Accessing Aggregates in Filters and Weights

Any DB access in a filter or weight slows down the scheduler. Until the end of kilo, there was no way to deal with the scheduler accessing information about aggregates without querying the DB in every call to host_passes () in a filter.

3. Filter Scheduler Alternatives

For certain use cases, radically different schedulers may perform much better than the filter scheduler. We should not block this innovation. It is unreasonable to assume a single scheduler will work for all use cases. However, to enable this kind of innovation in a maintainable way, a single strong scheduler interface is required.

4. Project Scale issues

There are many interesting ideas for new schedulers, like the solver scheduler, and frequent requests to add new filters and weights to the scheduling system. The current nova team does not have the bandwidth to deal with all these requests. A dedicated scheduler team could work on these items independently of the rest of nova. The tight coupling that currently exists makes it impossible to work on the scheduler in isolation. A stable interface is required before the code can be split out.

3.8 Future Scheduler Enhancements

Following enhancements are being contemplated to make the process of scheduling the resources more effective under OPEN STACK

1. Fixing the Scheduler DB model

We need the nova and scheduler data models to be independent of each other. The first step is breaking the link between the Compute Node and Service DB tables. In theory where the Service information is stored should be pluggable through the service group API, and should be independent of the scheduler service. For example, it could be managed via zookeeper rather than polling the nova DB. There are also places where filters and weights call into the nova DB to find out information about aggregates. This needs to be sent to the scheduler, rather than reading directly from the nova database.

2. Versioning Scheduler Placement Interfaces

At the start of kilo, the scheduler is passed a set of dictionaries across a versioned RPC interface. The dictionaries can create problems with the backwards compatibility needed for live-upgrades. Luckily we already have the "oslo.versionedobjects" infrastructure we can use to model this data in a way that can be versioned across releases.

3. Sending host and node stats to the scheduler

Periodically nova-compute updates the scheduler state stored in the database. We need a good way to model the data that is being sent from the compute nodes into the scheduler, so over time, the scheduler can move to having its own database. This is linked to the work on the resource tracker.

4. Updating the Scheduler about other data

For things like host aggregates, we need the scheduler to cache information about those, and know when there are changes so it can update its cache. Over time, its possible that we need to send cinder and neutron data, so the scheduler can use that data to help pick a nova-compute host.

5. Resource Tracker

The recent work to add support for NUMA and PCI pass through has shown we have no good pattern to extend the resource tracker. Ideally we want to keep the innovation inside the nova tree, but we also need it to be easier. This is much related to the effort to re-think how we model resources, as covered by discussion about resource providers.

6. Parallelism and Concurrency

The current design of the nova-scheduler is very racy, and can lead to excessive numbers of build retries before the correct host is found. The recent NUMA features are particularly impacted by how the scheduler works. All this has led to many people running only a single nova-scheduler process configured to use a very small green thread pool.

The work on cells v2 will mean that we soon need the scheduler to scale for much larger problems. The current scheduler works best with less than 1k nodes but we will need the scheduler to work with at least 10k nodes.Various ideas have been discussed to reduce races when running multiple nova-scheduler processes. One idea is to use two-phase commit "style" resource tracker claims. Another idea involves using incremental updates so it is more efficient to keep the scheduler's state up to date, potentially using Kafka.

3.9 Current Contributions related to enhancements to the scheduler

Many New Algorithms have been presented in the Literature for implementing advanced schedulers within OPEN STACK. Some of the algorithms presented in the Literature include Fair Share Algorithm, V Sphere DRS with Nova, Round Robin, First in First out, Multi-objective Ant Colony Optimization, First Fit, Least residue, Ant Colony Optimization, Network aware placement etc. All the algorithms aim at optimizing the utilization of the resources. The constraints imposed due to delay in execution time of the tasks have not been considered. Even the issues of scheduling the VM Migrations have also not been much discussed in the literature.

IV. CONCLUSIONS

Resource scheduling, JOB scheduling and VM migration scheduling are the most important tasks that must be undertaken most efficiently so that SLA are adhered to. The issue of scheduling should be core of any cloud computing architecture. OPEN STACK is the platform available for researchers to investigate the research finding and come out with much improvement which are absolutely required to come out with high efficient cloud computing system. The understanding of how scheduling is implemented within OPEN STACK is required so that new algorithms can be implemented and efficiencies of introducing such new algorithms can be recognized. In this paper, complete description of how scheduling of resources is done under OPEN stack has been presented. Future extensions that can be explored have also been presented in this paper.

REFERENCES

1. Oleg Litvinski and Abdelouahed Gherbi, Experimental Evaluation of OpenStack Compute Scheduler , Oleg Litvinski and Abdelouahed Gherbi, Procedia Computer Science 19 (2013) 116 – 123 page no 117-122
2. Sisu Xi, Chong Li, Chenyang Lu, Christopher D. Gill, RT-OpenStack: CPU Resource Management for Real-Time Cloud Computing, pageno 1-83.
3. Lianhao Lu, Yingxin Cheng, Utilization-based Scheduling in OpenStack* Compute (Nova), September 21, 2015, Document Number: 332369-001, pageno 1-11
4. U.Vignesh, R.V Nataraj, Investigation on Openstack Nova Scheduler for Customizing Enterprise Private Cloud, International Journal for Research in Applied Science & Engineering Technology (IJRASET), Volume 4 Issue I, January 2016, ISSN: 2321-9653,page no 394-396
5. Feng-Jun Shang, An Initial Resource Scheduling Algorithm for OpenStack, 2017 Asia-Pacific Engineering and Technology Conference (APETC 2017), ISBN: 978-1-60595-443-1,pageno1394-1399
6. Haibo Li1, Yaodong Cheng, Integration of Openstack cloud resources in BES III computing cluster, IOP Conf. Series: Journal of Physics: Conf. Series 898 (2017) 062033 doi :10.1088/1742-6596/898/6/062033, pageno 1-6
7. Mohan Krishna Varma Nandimandalam, The VM Weighted Filter Scheduling Algorithm for OpenStack Cloud, Springer Nature Singapore Pte Ltd. 2017,
8. J.J. (Jong Hyuk) Park et al. (eds.), Advanced Multimedia and Ubiquitous Engineering, Lecture Notes in Electrical Engineering 448, DOI 10.1007/978-981-10-5041-1_52 pageno 308-313
9. Sanhita Sarkar, Western Digital Corporation, Research and optimization of OpenStack virtUAL MACHINE RESOURCE SCHEDULINGtechnology, pageno 3-12
10. Michael Scharf, Manuel Stein, Network-aware Instance Scheduling in OpenStack , 978-1-4799-9964-4/15/$31.00 ©2015 IEEE, pageno 1-6
11. Bezawada, A., Marella, S.T., Gunasekhar, T., A systematic analysis of load balancing in cloud computing, International Journal of Simulation: Systems, Science and Technology ,19(6), pp. 4.1-4.7,2018
12. Rathod, S.B., Reddy, K., Predictive virtual machine placement in decentralized cloud environment, ICIC Express Letters, 12(9), pp. 863-870,2018
13. Praveen, S.P., Rao, K.T., Janakiramaiah, B., Effective Allocation of Resources and Task Scheduling in Cloud Environment using Social Group Optimization, Arabian Journal for Science and Engineering,43(8), pp. 4265-4272,2018
14. Sultanpure, K.A., Gupta, A., Reddy, L.S.S., An efficient cloud scheduling algorithm for the conservation of energy through broadcasting, International Journal of Electrical and Computer Engineering,8(1), pp. 179-188,2018
15. Abhinav Chand, N.V., Hemanth Kumar, A., Marella, S.T. , Cloud computing based on the load balancing algorithm, International Journal of Engineering and Technology(UAE),7(4.7 Special Issue 7), pp. 131-135,2018
16. Rathod, S.B., Krishna Reddy, V., Decision making framework for decentralized virtual machine placement in cloud computing, International Journal of Engineering and Technology(UAE) ,7, pp. 705-709,2018
17. Jena, S.R., Rao, L.S., A study on energy efficient task scheduler over three-tier cloud architecture using green cloud, Journal of Advanced Research in Dynamical and Control Systems,9(18 Special Issue), pp. 1-9,2017
18. Balaji, K., Sai Kiran, P., Efficient resource allocation algorithm with optimal throughput in cloud computing, Journal of Advanced Research in Dynamical and Control Systems,9, pp. 1902-1910,2017
19. Potluri, S., Rao, K.S., Quality of service based task scheduling algorithms in cloud computing, International Journal of Electrical and Computer Engineering,7(2), pp. 1088-1095,2017
20. Phanikumar, V., Satyanarayana, K.V.V., Advanced data sharing and group scheduling procedure for dynamic resource allocation of cloud computing, Journal of Advanced Research in Dynamical and Control Systems,9(Special Issue 6), pp. 396-409,2017
21. Siva Nageswara Rao, G., Srinivasu, S.V.N., Hybrid approach for task scheduling in heterogeneous cloud based systems, Journal of Advanced Research in Dynamical and Control Systems,9(Special Issue 14), pp. 618-625,2017
22. Jena, S.R., Vijayaraja, V., Sahu, A.K., Performance evaluation of energy efficient power models for digital cloud, Indian Journal of Science and Technology,9(48),105639,2016
23. Jena, S.R., Vijayaraja, V., Sahu, A.K., Performance evaluation of energy efficient power models for digital cloud, Indian Journal of Science and Technology,9(48),105639,2016
24. Reddy, V.K., Deva Surya, K., Sai Praveen, M., (...), Vishal, A., Akhil, K, Performance analysis of load balancing algorithms in cloud computing environment, Indian Journal of Science and Technology,9(18),2016
25. Durga Lakshmi, R., Srinivasu, N.., A dynamic approach to task scheduling in cloud computing using genetic algorithm, Journal of Theoretical and Applied Information Technology,85(2), pp. 124-135,2016

AUTHORS PROFILE

I Uma Maheswara Rao is a Scholar persuing Ph.D under the guidance of Dr.JKR SASTRY and specializes in the area of Cloud Computing

Dr. JKR Sastry a double doctorate in CSE ,ECE and Management works for KLEF university as a Professor. He has published more than 245 papers which are indexed into Scopus, SCI and Google

Table -1 Sample List of resource Partitions

Serial Number	HOST Name	Image	IP Address	Size	Status	Task	Power State	Up Time
1.	COMP-1	Image-1	192.168.252.1	2 CPU | 512MB | 1.0GB	Error	None	No State	0 Minutes
2.	COMP-2	Image-1	192.168.252.2	2 CPU | 1024MB | 2.0GB	Active	None	Running	2 Minutes
3.	COMP-2	Image-1	192.168.252.2	2 CPU | 512MB | 1.0GB	Active	None	Running	123 Minutes
4.	COMP-1	Image-1	192.168.252.1	2 CPU | 1024MB | 1.0GB	Active	None	Running	340 Minutes
5.	COMP-2	Image-1	192.168.252.2	2 CPU | 512MB | 1.0GB	Active	None	Running	20 Minutes
6.	COMP-1	Image-1	192.168.252.1	2 CPU | 512MB | 1.0GB	Active	None	Running	15 Minutes
7.	COMP-2	Image-1	192.168.252.2	2 CPU | 512MB | 1.0GB	Active	None	Running	180 Minutes