VOtus: A Flexible And Scalable Monitoring Framework for Virtualized Clusters

M. Suhail Rehman, Mohammad Hammoud, Majd F. Sakr
Carnegie Mellon University in Qatar
Education City, Doha, State of Qatar
Email: {suhailr,mhhammou,msakr}@qatar.cmu.edu

***Abstract*—Large-scale distributed processing frameworks such as Hadoop are currently enjoying wide popularity for big data computation. Performance Analysis and monitoring under these frameworks are inherently difficult especially in a virtualized environment. Existing distributed monitoring tools can only report virtual resource usage. Such reported information might be insufficient for developers and users to acquire deep insights into the performance of distributed applications. This paper describes VOtus, an initial step towards extending the Otus monitoring tool for virtualized clusters on private clouds. By collecting supplementary metrics from the hypervisor (when available), VOtus allows users to effectively and efficiently monitor a virtualized cluster. It also provides enhanced comprehension of distributed applications, which helps in answering performance related queries that relate to capacity planning, placement, and migration of virtual machines on the cluster.**

I. Introduction

Organizations around the world are dealing with unprecedented amounts of data in today's age. To deal with the complexities of processing big data, distributed frameworks such as Hadoop [1] have become massively popular. Companies such as Yahoo! and Facebook use these frameworks extensively in production environments and process petabytes of data on a daily basis. For smaller, short term data processing requirements, such distributed frameworks can be deployed on virtualized infrastructure, thereby allowing better utilization of hardware resources. For instance, Amazon's Elastic MapReduce service [2] manages a user's EC2 virtual machines and automatically deploys Hadoop MapReduce jobs.

Analysis and performance characterization of applications that run on these frameworks is difficult due to the distributed nature and scale of the computing platform. Cluster services such as job handling, data distribution and file system services run as processes and are co-located with user jobs. For example, in Hadoop MapReduce, services such as *TaskTrackers* and *DataNodes* are co-located with map and/or reduce tasks, and run simultaneously as java processes. Resources are not exclusively allocated to a single job, making application performance analysis even harder.

Many tools have been developed to assist developers and cluster administrators to monitor, debug and analyze these frameworks [4], [5]. One example is Otus [3], a simple yet powerful tool that can collect metrics from thousands of nodes and perform resource attribution, as well as provide a flexible visualization interface to users. Otus can collect metrics at various levels in the cluster software stack, including the OS and some high-level services such as HDFS or Hadoop MapReduce. Figure 1 shows a sample view from the Otus tool, attributing different MapReduce processes with the amount of memory consumed on a single cluster node.

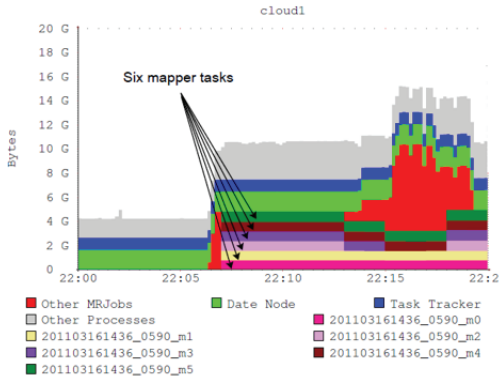


Fig. 1. Example of Otus monitoring the memory consumption of a node running a Hadoop Job (Figure from [3]).

Otus was designed to monitor and attribute resources on clusters that run on traditional bare metal hosts. When Otus is deployed on virtual machine hosts that form a virtualized cluster, it can only perform resource attribution of applications on the virtual resources within the VM. Ideally, Otus would be most useful if it could attribute applications running on virtual machines to the physical resources of the bare metal. As a first step towards this goal, we've developed *VOtus* by extending Otus to collect and present metrics from both the hypervisor layer along with metrics from within the VMs. To our knowledge, this is the first monitoring tool that collects and presents metrics for performance analysis from both layers. As a result we offer a consolidated visualization of physical and virtual resources, and provide developers and users with deeper insights into the performance of their applications.

II. Background

Contemporary monitoring systems such as [4], [5], [6] share many similarities with Otus. Figure 2 illustrates the typical architecture of a scalable monitoring system designed for large-scale clusters.

A monitoring system typically consists of client-side *collection* agents, *aggregator* agents, a *storage back-end*, and a *front-end application* (typically a GUI or a web-based

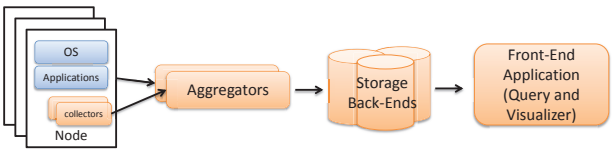


Fig. 2. Architecture of monitoring systems.

application). A collector is a background process that runs on each cluster host (or VM). It periodically reads performance metrics collected from the OS and/or the applications (e.g., CPU utilization or disk I/O operations per second) and passes them to a certain aggregator. An aggregator receives data from multiple collectors and makes decisions on where and how to store the data. In systems such as Ganglia [4], an aggregator also performs data-thinning, where historical data is compressed by aggregating the values of the collected metrics. Data is typically thinned at granularities of hour, day, week, etc. The storage back-end provides a central repository for storing and querying the metrics received from aggregators. Finally, the front-end application is used by users to query metrics and visualize them over a particular time range (e.g. the past hour or day etc.).

Although many of the available distributed monitoring systems enable cluster management, they fail in providing necessary information to answer fundamental questions (e.g., what is the bottleneck resource being exhausted by a user's application). Otus addresses such deficiencies and effectively facilitates performance analysis of distributed data-intensive applications. However, when deployed in a virtualized environment, Otus does not help much in understanding resource requirements of these applications.

A. Architecture of Otus

The Otus monitoring tool is a flexible and scalable resource monitoring system for data-intensive cluster systems. The novelty of Otus lies in the following:

- It can store metrics persistently without thinning the resolution of data over time. This allows users to analyze and debug distributed applications more effectively even after completion.
- It can attribute resources to specific applications, jobs and services. This allows users to identify resource bottlenecks and scrutinize their application requirements.

Component	Implementation
Collector	tcollector with Otus plugins
Aggregator	time-series daemon (TSD)
Storage Back-End	HBase
Front-end Application	Django-based webapp

TABLE I
VARIOUS COMPONENTS OF OTUS.

Table I lists all Otus's components. Otus uses OpenTSDB [6] to handle collecting, aggregating and storing data. OpenTSDB is a scalable time-series database implementation over HBase [7] developed at StumbleUpon. We now describe each of Otus's components in detail.

a) Collection: Otus relies on the *tcollector* [8] application existing in OpenTSDB for data collection. tcollector utilizes user-written scripts to produce output in a specific format to standard output. It automatically launches collectors on behalf of the user, reads recorded metrics and sends them to designated aggregators (known as *time-series daemons* or TSDs in OpenTSDB's parlance). Default tcollector plugins that come with OpenTSDB collect overall CPU, memory and disk utilization metrics. Plugins for Otus collect information resource usage at the process level, obtained by parsing files in the `\proc` directory. Otus also tags the metrics by process, and can identify each as being a generic java process, a map/reduce task or a Hadoop daemon.

b) Aggregation: Otus uses OpenTSDB's Time-Series Daemon (TSD) aggregator. TSD is a server daemon that actively listens on a user-specified TCP port for metrics that are dispatched by tcollector daemons running on client nodes. TSD receives and validates incoming metrics and forwards them to HBase for storage.

c) Storage Back-end: Incoming data points need to be stored in a persistent and scalable manner. When running Otus on hundreds or possibly thousands of nodes, a robust and distributed storage back-end will be required. Otus and OpenTSDB use the HBase distributed database system from Apache in order to persistently store metrics for large clusters.

d) Front-end Application: The Otus front-end consists of a web application that allows users to view particular clusters/nodes and running MapReduce jobs. It also allows users to create custom queries over the range of metrics that are recorded by OpenTSDB.

B. Limitations of Otus on Virtualized Clusters

When Otus is deployed on a virtualized cluster, the collectors are run on virtual hosts instead of physical ones. The information polled by these collectors will be from the guest OSs on the VMs and will reflect the usage of virtual rather than physical resources. This does not help users in finding out the exact resource requirements of their distributed applications. If Otus is to be effective for virtualized clusters, it should access physical as well as virtual resource metrics. This will be especially useful in answering performance related queries such as capacity planning, placement, and migration of VMs on the cluster.

Virtualization-related metrics are typically obtained through hypervisors such as VMWare's ESX [9] and Xen [10]. Command-line tools such as esxtop or xentop can be used for real-time monitoring and analysis of VMs but are cumbersome for analysis after-the-fact. Additionally, if users need to cut across abstraction levels and analyze applications that are running on VMs and would like to see their effect on physical hardware, they would need to use two tools - one that monitors processes within VMs and another that monitors VMs and resources from the hypervisor.

III. DESIGN OF VOTUS

VOtus has been designed with an aim to keep most of the original architecture of Otus intact. Otus is robust and flexible

enough to incorporate additional metrics.

A. Collecting performance metrics from the Hypervisor

A number of Application Programming Interfaces (APIs) provide 'hooks' which allow users to not only query performance metrics and get status updates, but also to manage VMs (e.g., by dynamically creating, destroying or migrating VMs across physical hosts). While a number of APIs are available that target specific hypervisors, the Red-Hat backed libvirt [11] project aims to provide a common API for most popular hypervisors such as Xen, KVM, VMWare, OpenVZ, among others. Currently, libvirt has sparse support for some hypervisors (e.g., VMWare ESX).

For the purpose of developing VOtus given our infrastructure (see Section IV), we used VIJava [12], an open-source API which targets solely the VMWare hypervisor. VIJava is a rich API and exposes hundreds of metrics on a VMWare-based cluster. Nonetheless, the techniques we describe in this section are general enough to be applied to APIs such as libvirt in order to monitor other virtualized systems as well.

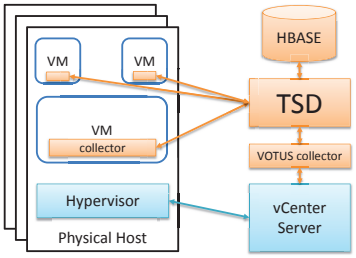


Fig. 3. Design of VOtus.

B. VOtus Design

Figure 3 shows the design of VOtus. VMware organizes physical hosts into *clusters*. In order to manage a cluster effectively, VMWare offers a product called vCenter that allows managing all available physical hosts and the VMs resident on them. The vCenter server also hosts a VMWare web service which allows remote hosts to connect over HTTPS and perform monitoring or management functions. We used the VIJava SDK to write plugins that connect to the vCenter server and query any required hypervisor performance metric. All OpenTSDB metrics must be of the form `<metric name><timestamp><value>[tags]`. Our plugins comply to such a format and output metrics that can be collected and shipped to the OpenTSDB back-end of Otus. Table II provides a list of some interesting metrics that we poll from the hypervisor.

IV. Two Case Studies

We deployed VOtus on a 14-node private cloud running VMware vSphere 4.1. One of the VMs was configured with the VOtus aggregator and storage back-end, while collectors were deployed on the physical hosts and VMs. The following case studies illustrate the usage of VOtus when running distributed applications on a 4-VM Hadoop cluster. Figure 4 illustrates the setup. The 4 VMs were provisioned on 2 physical hosts,

Metric Name	Description
vmm.physical.cpu.usage.percent	Utilization of Host CPU
vmm.physical.cpu.hwthreads.num	Total number of Hardware Threads on Host Machine)
vmm.physical.memory.total.mb	Total Memory on Physical Host
vmm.physical.memory.consumed.bytes	Memory consumed in MB by VMs on physical Host
vmm.physical.memory.active.bytes	Physical Memory actively used by VMs on a physical host

TABLE II
List of some of the metrics in VOtus.

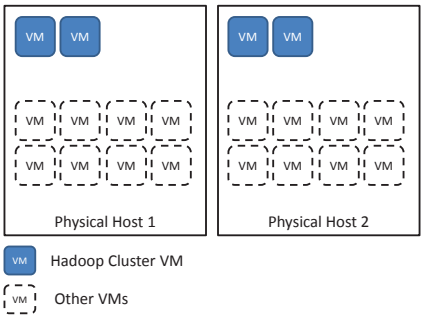


Fig. 4. VM setup for the two case studies.

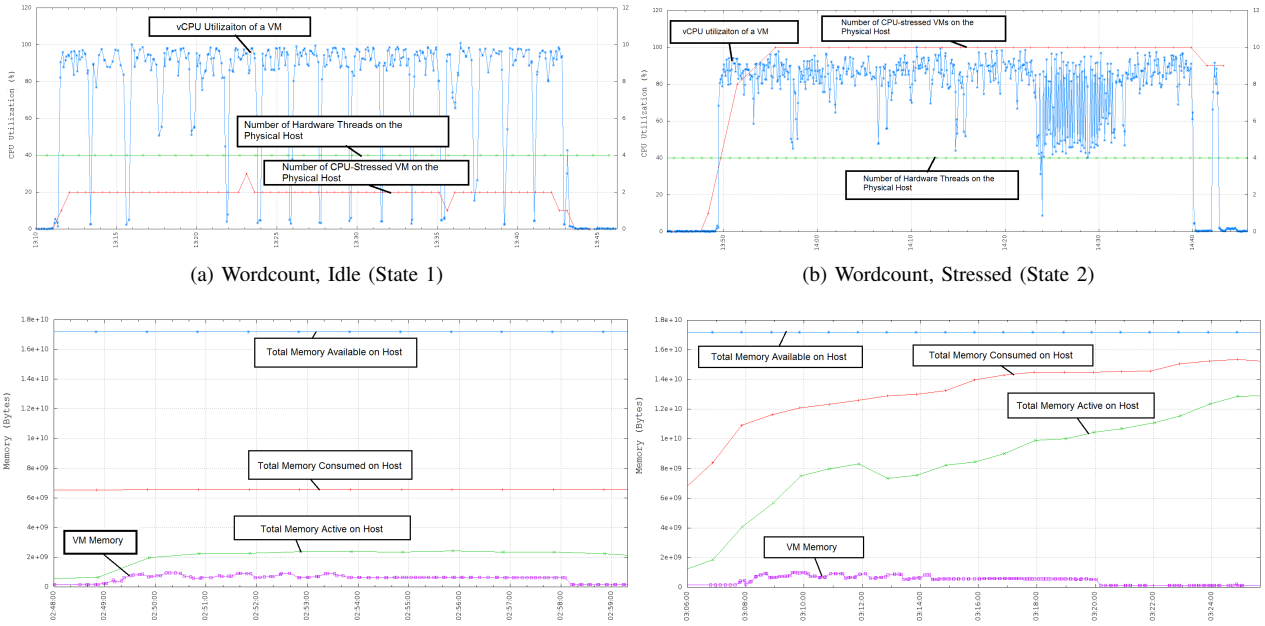
with 2 VMs on each. There are other VMs that have been also provisioned on the 2 physical hosts. Our objective is to measure the effect of co-located VMs on the performance of jobs running on the 4-VM cluster. So synthesize co-located VM load, we used a stress application [13], which we setup to execute artificial workloads on all of the other VMs residing on physical host 2. Finally, for each case study, we run a Hadoop job in two states:

1) **(State 1)** Job executed when other VMs are idle.
2) **(State 2)** Job executed when all other VMs on one of the physical hosts are stressed for a resource.

A. Monitoring CPU Load

For our first case, we run the *Wordcount* Hadoop job on 8 GB of random text using 4 VMs. The runtime of the Hadoop job in state 2 is roughly 50% slower than state 1, since wordcount is a CPU-bound Hadoop job and half the VMs in state 2 are CPU-starved owing to the other VMs that are stressing the CPU.

In Figures 5a and 5b, we plot the output of VOtus monitoring the second physical host, as well as one of the VMs on that host. From the VM, we plot the vCPU utilization - this is the information provided by Otus. From the physical host, we plot the number of physical hardware threads available and the number of VMs that have high CPU load (which we arbitrarily define as 80% or higher), obtained from the hypervisor using our VOtus plugins. In both figures, we see that the VMs reach near 100% vCPU utilization. In Figure 5a, only 2 VMs (the ones running the Hadoop job) have a CPU load of 80%, which is below the number of hardware threads available on the physical host. This indicates that there is available CPU capacity on this physical host and additional VMs could be provisioned or migrated to this host for this particular job.



(a) Wordcount, Idle (State 1)

(b) Wordcount, Stressed (State 2)

(c) Hadoop Sort, Idle (State 1)

(d) Hadoop Sort, Stressed (State 2)

Fig. 5. Case Studies

However, in Figure 5b, we have a maximum of 10 VMs that are experiencing high CPU load, as reported by the hypervisor. Such additional information provided by VOtus grants us a clearer picture of the actual utilization of the physical CPU on that host. The plot clearly indicates that this physical host is currently over-utilized with respect to CPU. A straightforward solution to manage the observed performance deterioration is to migrate some of the VMs away from this over-utilized physical host.

B. Monitoring Memory Usage

For our second case, we run the *Sort* job on 4 GB of random binary data using 4 VMs. The execution time of the Hadoop job in state 2 is roughly 50% slower than state 1, since sort is a memory-bound Hadoop job and half the VMs in state 2 are memory-starved. In this case, Figures 5c and 5d show the usefulness of the hypervisor metrics collected by VOtus (i.e., total memory, consumed memory and active memory on host). In particular, we are able to indicate a memory stress situation as opposed to purely viewing the VM memory metrics. Clearly, by incorporating performance metrics from the hypervisor, VOtus is able to provide deeper insights into the actual resource usage on the physical hosts.

V. Conclusion

We have presented VOtus, a flexible and scalable framework for monitoring virtualized clouds. VOtus provides access to performance metrics that are collected from within VMs and the hypervisor, thereby allowing users to analyze application performance from multiple viewpoints. VOtus is an initial step towards adapting Otus for virtualized clusters. We foresee VOtus being a useful monitoring tool for researchers in virtualized data-intensive centers. In the future, we plan to work on additional metrics, such as those pertaining to disk and network usage in virtualized environments. We also plan to work on resource attribution under virtualized clusters.

Acknowledgments

The authors would like to thank the entire Otus team, Kai Ren, Julio López, and Garth Gibson for their invaluable support in setting up Otus on our system and their essential feedback on this piece of work. We are also very grateful to Jim Gargani for his assistance with this project.

References

[1] Apache hadoop. http://hadoop.apache.org/
[2] Amazon elastic mapreduce. http://aws.amazon.com/elasticmapreduce/
[3] K. Ren, J. López, and G. Gibson, "Otus: resource attribution in data-intensive clusters," in *Proceedings of the second international workshop on MapReduce and its applications*, ser. MapReduce '11. New York, NY, USA: ACM, 2011.
[4] M. L. Massie *et al.*, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, 2004.
[5] A. Rabkin and R. Katz, "Chukwa: A system for reliable large-scale log collection," in *Proceedings of the 24th international conference on Large installation system administration*. USENIX Association, 2010.
[6] OpenTSDB. http://opentsdb.net/
[7] (2011) Apache HBase. http://hbase.apache.org/
[8] StumbleUpon. tcollector for opentsdb. http://opentsdb.net/tcollector.html
[9] VMware ESX hypervisor. http://www.vmware.com/products/vsphere/esxi-and-esx/index.html
[10] P. Barham *et al.*, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, 2003.
[11] (2011) libvirt - virtualization api. http://libvirt.org/
[12] S. Jin. Vmware vi (vsphere) java api. http://vijava.sourceforge.net/
[13] Amos Waterland. stress - a workload generator for POSIX systems. http://weather.ou.edu/ apw/projects/stress/