2013 IEEE International Conference on Cloud Computing Technology and Science

Varanus: In Situ Monitoring for Large Scale Cloud Systems

Jonathan Stuart Ward and Adam Barker
School of Computer Science
University of St Andrews, UK

***Abstract*—Monitoring is an essential aspect of maintaining and developing computer systems which increases in difficulty proportional to the size of the system. The need for robust monitoring tools has become more evident with the advent of cloud computing. Infrastructure as a Service (IaaS) clouds allow end users to deploy vast numbers of virtual machines as part of dynamic and transient architectures. Current monitoring solutions, including many of those in the open-source domain, rely on outdated concepts including manual configuration and centralised data collection and adapt poorly to membership churn. In this paper we propose the development of a cloud monitoring system to provide scalable and robust lookup, data collection and analysis services for large-scale cloud systems. In lieu of centrally managed monitoring we propose a multi-tier architecture using a layered gossip protocol to aggregate monitoring information and facilitate lookup, information collection and the identification of redundant capacity. This allows for a resource aware data collection and storage architecture that operates over the system being monitored. This in turn enables monitoring to be done in situ without the need for significant additional infrastructure to facilitate monitoring services. We evaluate this approach against alternative monitoring paradigms and demonstrate how our solution is well adapted to usage in a cloud-computing context.**

I. Introduction

Cloud computing has become the premier means to rapidly deploy internet scale systems. Amazon Web Services (AWS), the most prevalent public cloud provider, has become the worlds largest internet host [5] and cloud computing has become a standard, well accepted, paradigm. Central to the concept of cloud computing is the idea of rapid elasticity: the mechanism to provision, scale and terminate infrastructure in a short amount of time. Despite the intrinsic importance of rapid scalability it is poorly catered for by many applications built upon the cloud. Many applications used on the cloud are mired in concepts and technologies which were never designed to tolerate rapid elasticity at scale.

Among the applications which have seen insufficient adaption towards cloud environments are monitoring tools. Within the cloud domain there is a clear dichotomy of monitoring tools. There is either the choice to deploy one's own monitoring infrastructure at ones own cost or to leverage a monitoring service which abstracts the monitoring infrastructure away from the user. In either case monitoring is achieved through the use of a pool of monitoring servers which collect information from a series of monitored hosts. As the size of the system changes the pool of monitoring scales to meet demand. This inevitably results in substantial overhead costs for large scale systems. In the case of many common monitoring systems this issue is further compounded by the need for manual configuration and an intolerance to churn which make monitoring a laborious process. The lack of user deployable monitoring systems which are well adapted for use in a cloud environment is a limiting factor in provisioning cloud architectures and is restrictive in private clouds whereby many proprietary monitoring services are unavailable.

We therefore propose Varanus[1] a highly scalable decentralised monitoring system for cloud computing. In this paper we propose and evaluate Varanus as an architecture for performing fully decentralised data collection, analysis and monitoring of cloud virtual machines (VMs). Our approach rejects conventional monitoring concepts in lieu of a decentralised approach based upon a layered gossip algorithm [1]. By leveraging this mechanism we propose an architecture which is highly scalable, and attempts to eliminate the need for additional, dedicated monitoring infrastructure. Through this approach we intend to meet the requirements imposed by rapid-elasticity [9] and provide a monitoring system suitable for cloud computing.

II. Architecture

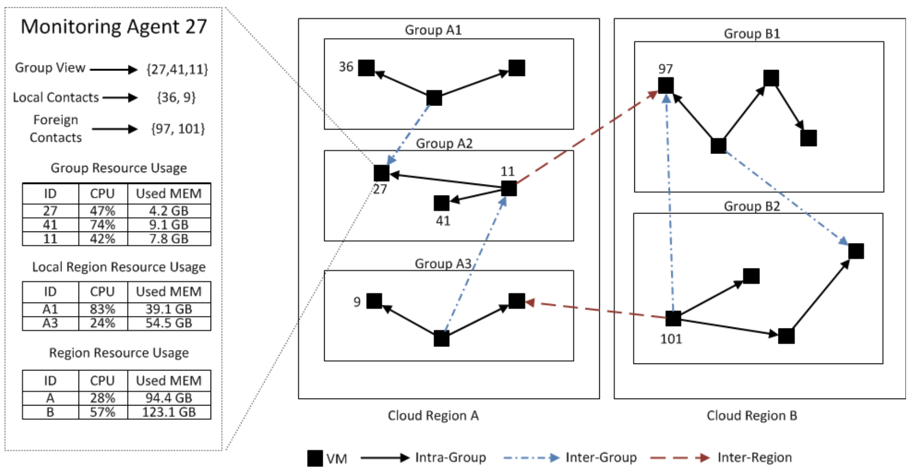
A. Layered Gossip Broadcast

A probabilistic broadcast protocol otherwise known as gossip or epidemic protocol is the primary communication mechanism employed by Varanus. In large scale cloud systems individual VMs operate under a range of computation and communication constraints. By distributing the computational complexity of an operation over the system, gossip protocols offer a means to develop mechanisms better suited to large scale systems. Gossip protocols have been demonstrated to be effective mechanisms for providing robust and scalable services for distributed systems including information dissemination [2], aggregation [3] and failure detection [6].

The basic operation of the Varanus gossip protocol consists of the periodic, pairwise propagation of state between two processes. This mechanism underpins the data collection and agreement protocols which support monitoring functions. Each monitoring agent participates in a gossip based overlay network. Using this overlay monitoring agents propagate and receive state from other, nearby, agents. This is achieved

[1]Varanus is the genus of the monitor lizard

978-0-7695-5095-4/13 $31.00 © 2013 IEEE
DOI 10.1109/CloudCom.2013.164



Group Resource Usage

ID	CPU	Used MEM
27	47%	4.2 GB
41	74%	9.1 GB
11	42%	7.8 GB

Local Region Resource Usage

ID	CPU	Used MEM
A1	83%	39.1 GB
A3	24%	54.5 GB

Region Resource Usage

ID	CPU	Used MEM
A	28%	94.4 GB
B	57%	123.1 GB

Fig. 1. Inter and Intra Cloud Communication Model

by performing a pull-push operation with neighbouring correspondents. Correspondents exchange state every $T_{interval}$ seconds according to the following scheme:

1) Select a number of targets from the group view equal to the $Fanout$ value.
2) Push entries from the local state and pull entries from the targets state.
3) The agent may now have duplicate state. For each instance of duplicate state, the oldest is removed.

Therefore the rate of dissemination of data from a single process to all other processes can be described by the following equation:

$$S_{t+1} = T_{interval} \times Fanout \times \frac{S_t X_t}{n} \tag{1}$$

where S is the number of susceptible processes (those which have not yet received the information) , X is the number of infected processes (those which have received the information), n is the number of processes and t is the current timestep. Therefore, the delay in propagating information can be greatly reduced by decreasing the interval at which communication occurs (thus increasing the frequency) and by increasing the fanout value (thus increasing the number of correspondents selected as targets). Fanout in Varanus is by default $\log(groupsize)$ but this can be altered. The frequency is calculated according to the rate at which monitored values change at.

In addition to this mechanism, preferential target selection is used to reduce the delay in propagating state. Targets are selected based on a weighting scheme which uses round-trip time estimates in order to select targets which are topologically closer. Each round of gossip is spatially weighted according to the scheme proposed in [4], using RTT as a distance metric in order to propagate updates to all nodes within distance d within $O(\log^2 d)$ time steps.

This scheme results in increased memory usage and constant background communication but achieves rapid state propagation and resilience to churn and failure. Within a cloud where there is high bandwidth, low latency and no service metering this trade-off is acceptable.

B. Communication Hierarchy

Varanus employs a layered gossip approach in order to reduce the time required to disseminate information and to exploit the differing network properties in and between cloud regions. The rationale for this hierarchy is rooted in the differences between intra and inter cloud communication. Within clouds there is high bandwidth, low latency and the connection is unmetered. This is true of virtually all commercial cloud providers. It is also true of any private cloud with a public network between cloud regions. This environment lends itself to the use of rapid information dissemination based on an unreliable protocol such as UDP. Between cloud regions this is not as feasible as costs arising from latency and bandwidth metering force communication to be performed in a slower, more reliable fashion. This therefore requires a slower, reliable protocol to synchronise state between regions.

The gossip protocol described in section A, is applied at every level of the hierarchy. What differs between each level is the information which is communicated and the frequency at which communication occurs. There are three levels of the hierarchy as shown in Figure 1:

1) Intra Group: communication between monitoring agents within the same group. This occurs at a near constant rate. Every time a state change occurs the correspondent propagates the state change to its neighbours. At this level of granularity, the full state stored by the monitoring agent is propagated to its neighbours.
2) Inter-Group: communication between monitoring agents in different groups within the same region. This occurs at a frequent but non constant rate. Periodically state is propagated to external groups according to a shifting interval. At this level, only aggregated values for the region resource usage and a small subset of local contacts and foreign contacts are propagated.

3) Inter-Region: communication between monitoring agents in different groups and different cloud regions. This occurs proportionally to the inter-group rate. At this level an aggregate value for the entire region and subsets of the local and foreign contacts are propagated between regions.

C. Stored State

The monitoring agent operates a state store which maintains a subset of global membership information, local and group monitoring state and aggregates of other monitoring state. In full, the state store maintains the following:

- Group View: a set of hosts within the same group.
- Local Contacts: a small, fixed size set of hosts within other groups within the same cloud region. Each entry carries fields for round-trip time and heartbeat count.
- Foreign Contacts: a small, fixed size set of hosts within other groups within other cloud regions.
- Group Resource Usage: a dictionary, proportional to the size of the group storing recent resource usage of group members including CPU, memory, disk and network usage.
- Local Region Resource Usage: set of aggregated values representing resource usage within each group within the same region.
- Region Resource Usage: a set of aggregated values representing resource usage within a region.
- Publisher Functions: a set of data collection functions which populate the state store.
- Analysis Functions: a set of analysis functions which operate on designated analysis nodes.

This state is propagated according to the communication hierarchy described in B.

D. Data Collection

The set of resource usage metrics stored as part of the set is populated by a series of publisher functions. Each publisher runs in parallel and publishes a metric to the state store. Publishers are also stored within the monitoring agent's state store allowing for publishers to be added or removed at runtime. By default the monitoring agent includes publishers providing the following metrics: CPU usage, memory usage, network traffic and disk capacity.

Additional publishers can be provided when the monitoring agent is installed or at runtime. Publisher functions can be propagated within groups in the same manner as other state allowing for the user to introduce new functions to obtain additional metrics after initial deployment allowing for the programmatic alteration of the monitoring service. Publishers can be introduced as a one-time function in order to obtain values which are seldom required or as permanent additions which will continuously publish their respective metric. This allows the data that the system collects to be programatically altered enabling the monitoring system to alter its behaviour as requirements shift.

E. Group Allocation

Group allocation operates upon the premise that monitoring information from a VM is most pertinent to other VMs which are similar to the first. This is the case for many cloud use cases including load balancing, batch computing and any application benefiting from rapid elasticity. In order to group related VMs an algorithm reminiscent of the distributed k-nearest neighbour algorithm is used. Each VM computes a feature vector describing its software configuration. The feature vector details the OS and significant software which is the software that the VM has been specifically provisioned to run. This includes: web servers, database servers and compute nodes. When bootstrapping, VMs compute this feature vector while groups compute an average feature vector. The averaged vector describes the most common attributes of that group. The bootstrapping process utilises a k-nearest neighbour like algorithm to determine which, of the set of group feature vectors is closest to that of the new VM. The new VM then joins the group which has the greatest similarity to itself.

F. Data Analysis

Monitoring systems are typically required to fulfil a minimum of analysis functions in order to reduce human involvement with the monitoring process. In existing systems analysis is performed as a centralised function usually on a dedicated monitoring server. In the case of Varanus, analysis functions are distributed throughout the system. Each group is responsible for performing its own analysis.

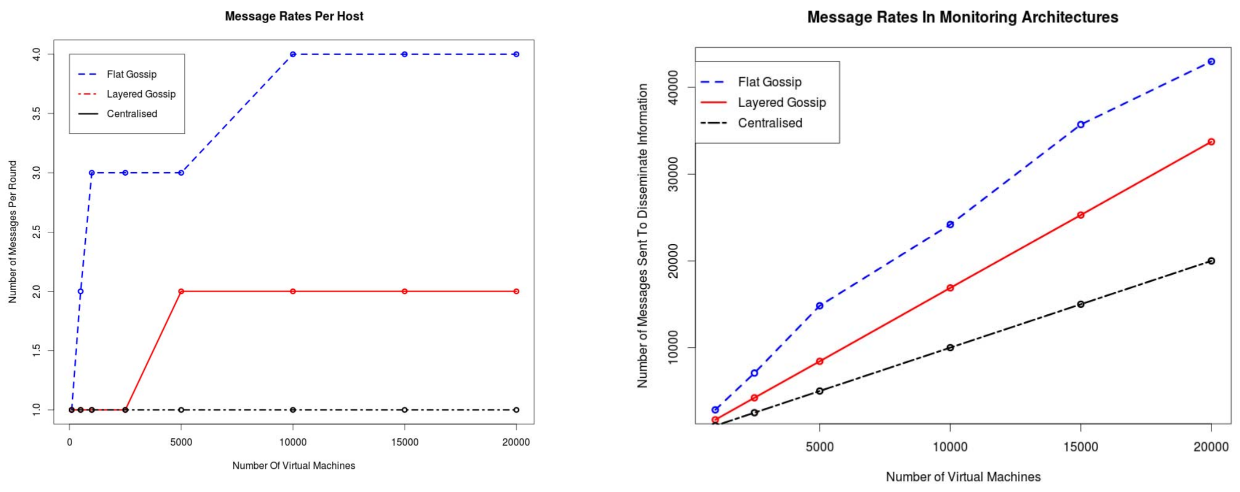
The propagation of resource usage information allows for the least utilised VMs within a group to be trivially identified. In the case where there are underutilised VMs, those VMs are nominated to perform analysis services for the group. Otherwise, analysis services are provided by the least loaded VMs within the group. If all VMs within the group are heavily loaded it is necessary for an additional dedicated analysis VM to be provisioned. Analysis functions can be introduced and propagated in the same manner as other state.

III. Evaluation

In the following section we will numerically evaluate Varanus against alternative monitoring strategies.

A common criticism of gossip protocols is their potentially significant use of bandwidth. In Varanus computational complexity is reduced at the expense of communication complexity. In order to examine the implications of this trade-off we simulated the layered gossip architecture of Varanus, a more basic flat gossip scheme and a conventional centralised monitoring architecture (such as that of Nagios). The simulation was created in Python using the Nessi library [7]. Our software simulation implements the same data collection strategy as the actual monitoring system. The simulation records the number of messages required in order to disseminate a monitoring metric from one host to the monitoring system. Figure 2 illustrates the findings from this experiment.

The two gossip based architectures have notably higher message rates than that of the centralised architecture. In the case of the flat gossip architecture the message rate is



(a) Simulated Messages Rates of Varanus and other architectures per host

(b) Simulated System Wide Message Rates of Varanus and other architectures

Fig. 2. Message rates in monitoring architectures

around three times that of the centralised architecture. The additional overhead is due to the number of messages required to aggregate and then propagate information throughout the system. Despite a greater message rate than the centralised collection scheme, Varanus has a relatively conservative rate when compared to the flat scheme. This is due to the grouping and layering mechanisms present in Varanus which enforce a communication hierarchy which limits global communication.

Despite being double the rate of the centralised system, a vast disparity only emerges when operating at scale. Even at scale we argue that the message rates imposed by Varanus are acceptable in a cloud environment. The high bandwidth, low latency environment present in clouds allows for applications to leverage greater message rates. The scare resource in cloud environments is CPU and memory and not bandwidth. We therefore contend that Varanus has achieved an acceptable level of background communication in exchange for decentralised monitoring.

IV. Conclusion

We have proposed Varanus, a highly decentralised monitoring system as a means to monitor large scale cloud systems without (or with a reduced need) for dedicated monitoring infrastructure. Varanus has significant benefits over existing systems, notably it provides mechanisms for programmatic runtime reconfiguration and executes monitoring analytics in a scalable, resource aware manner. As large cloud hosted systems become increasingly common we propose our system as a means of reducing the overhead, complexity and bottlenecks inherently associated with current monitoring technologies.

The architecture described here provides a mechanism for the scalable collection of monitoring metrics and the analysis of these metrics. It does not however provide a full, comprehensive monitoring suite. In order to fully monitor a system Varanus must become aware of applications running on the monitored system. What Varanus does provide however, is the foundation for an application aware monitoring system. The primary concern of future work is to develop application monitoring functions on top of the Varanus architecture described here.

References

[1] Ken Birman. The promise, and limitations, of gossip protocols. *SIGOPS Oper. Syst. Rev.*, 41(5):8–13, October 2007.

[2] Anwitaman Datta and Rajesh Sharma. Godisco: selective gossip based dissemination of information in social community based overlays. In *Proceedings of the 12th international conference on Distributed computing and networking*, ICDCN'11, pages 227–238, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, August 2005.

[4] David Kempe, Jon Kleinberg, and Alan Demers. Spatial gossip and resource location protocols. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 163–172, New York, NY, USA, 2001. ACM.

[5] Netcraft Ltd. http://news.netcraft.com/archives/2013/01/07/january-2013-web-server-survey-2.html, 2013.

[6] Robbert van Renesse, Yaron Minsky, and Mark Hayden. A gossip-style failure detection service. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Middleware '98, pages 55–70, London, UK, UK, 1998. Springer-Verlag.

[7] J. Vernez, J. Ehrensberger, and S. Robert. Nessi: a python network simulator for fast protocol development. In *Computer-Aided Modeling, Analysis and Design of Communication Links and Networks, 2006 11th International Workshop on*, pages 67–71, 2006.

[8] Spyros Voulgaris, Mrk Jelasity, and Maarten Van Steen. A robust and scalable peer-to-peer gossiping protocol. In *In 2nd Intl Workshop Agents and Peer-toPeer Computing, LNCS 2872*, pages 47–58. Springer, 2003.

[9] Jonathan Stuart Ward and Adam Barker. Semantic based data collection for large scale cloud systems. In *Proceedings of the fifth international workshop on Data-Intensive Distributed Computing Date*, DIDC '12, pages 13–22, New York, NY, USA, 2012. ACM.