Trustworthy and Resilient Monitoring System for Cloud Infrastructure*

S Padhy, D Kreutz
LASIGE, Faculty of Sciences
University of Lisbon, Portugal
{spadhy,kreutz}@lasige.di.fc.ul.pt

A Casimiro, M Pasin
LASIGE, Faculty of Sciences
University of Lisbon, Portugal
{casim,pasin}@di.fc.ul.pt

ABSTRACT

Current monitoring systems for cloud infrastructure run locally or are based on centralized model approaches such as HP Openview, Microsoft Operation Manager and ArcSight. Additionally, they do not look deep into resilience and delivering trustworthy data of its own services under crash or Byzantine failures caused by attackers or any other kind of sources. This work proposes a fault and intrusion tolerant (FIT) monitoring system for cloud computing infrastructures. We assume Byzantine failure model and use state machine replication for providing the trustworthy and resilient monitoring service.

1. MOTIVATION

Cloud computing is becoming popular for vast available computing resources on clouds at affordable price and hassle free installation. A typical data center deploys several machines to provide IaaS, SaaS and PaaS. Besides, it has control, monitor and management services to its internal network such as LDAP, DNS, DHCP, SMTP, SNMP, Puppet and CFEngine. These services are responsible for the correct operation of cloud infrastructure and its security. It is envisioned that cloud services are going to be used in public critical infrastructures such as grid, healthcare and other strategic applications. Concerns about Quality of Service (QoS) and Quality of Protection (QoP) when using IaaS for these environments become very critical because companies and people rely their services on these cloud infrastructures expecting 100% of uptime. Thus, they need to be monitored in a resilient and trustworthy way for security and availability.

A typical cloud infrastructure monitoring system comprises probes or agents connected to a console through a centralized system for event message gathering. One existing solution is ArcSight Enterprise Security Manager. It works in a centralized way collecting information, basically logs, from various devices and applications, and generating alerts on the occurrence of some critical events. HP Openview and Microsoft Operation Manager are examples of similar tools. Further, there are also some more specific IaaS monitoring tools such as Cloudkick, Zenoss, Amazon Cloudwatch, LogicMonitor and CloudSec. These monitoring tools also work in centrailzed model and do not ensure availability of its service, being a single point of failure or attack. To address this problem, as shown in Figure 1, we propose to design a FIT monitor with replication to ensure the trustworthiness and resiliency of the monitoring system itself. We use a topic-based event-based publish/subscribe system for event message dissemination. Probes become publishers while consoles act as subscribers. The FIT monitor works as a reliable framework for delivering data from publishers to subscribers. To ensure Byzantine fault tolerance in our proposal we use state machine replication. This makes the monitoring system resilient to any kind of failures, allowing system and network monitoring to be trustworthy.

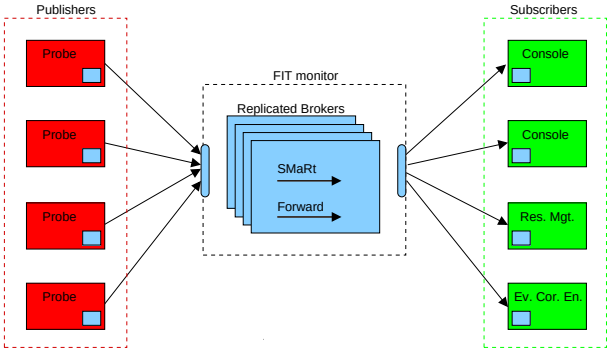

Figure 1: Abstraction of FIT monitoring system

2. SYSTEM MODEL

Probes (or agents), FIT monitor, and consoles are major components of the proposed architecture for monitoring IaaS. Further, we make the following assumptions for our system:

Network. All probes and consoles can reach each FIT monitor replica.

Synchrony. Asynchronous system model where messages can be arbitrarily delayed.

Faults. Byzantine failure/fault model where any of the FIT monitor replicas may crash or behave arbitrarily. The monitoring system can tolerate upto f faulty replicas. There can be event message delay or loss due to link failure or some malicious attacker. However, we assume eventually bounded message delay. To ensure integrity of the event message generated by using a probe, we assume that probes are able to digitally sign messages with its private key(e.g. RSA). With this approach the monitoring system itself, as well as subscribers, can verify the message integrity and authenticity by using the probe's public key.

3. PROPOSED FIT MONITORING SYSTEM

*This work is supported by the FCT, through project TRONE (CMU-PT/RNQ/0015/2009) and the Multiannual and CMU-Portugal programmes.

We explain the building blocks of the proposed FIT monitoring system and detail also its architecture. Fig 1 represents its abstract architecture while Fig 2 shows the internal composition of a broker.

3.1 Building Blocks

Event handling: We propose to use the Publish/Subscribe (P/S) paradigm for communication and event handling [3]. Probes, event brokers and consoles can be mapped to a publish/subscribe system. A probe publishes messages to event brokers. The FIT monitor forwards event messages to its subscribers in a trustworthy way. Consoles, resource managers and different applications subscribe themselves on the brokers to receive specific information. In this work, we focus on topic-based publish/subscribe. There are several works on search methods; topic-based, content-based and type-based, in P/S systems. However, only a few works addressed the fault-tolerance in P/S system. In [4], the authors provide a P/S algorithm that tolerates up to δ crash failures. However, as far as we know, Byzantine fault-tolerant event brokers have not been addressed in the literature. Our FIT broker design for Byzantine fault-tolerance will contribute to the P/S system research.

Client side: A probe is a program that generates a message on the occurrence of events such as security threats and system state changes like network flow statistics, storage space usage, and processor usage. A console is a software and/or hardware component that receives monitoring messages for displaying or alerting system administrators. Instead of a console, a subscriber may also be a resource manager or an event correlation engine. A console is also integrated with a voter interface. It performs a voting on messages received from different brokers, for the same event, before delivering the result to the console. So, even if a broker has been compromised it will not affect data correctness delivered to subscribers. Both probes and consoles are clients of the broker. Probes generate event messages which are pushed into the event broker through a previously defined event channel. On the other hand we have the consoles receiving messages from brokers. Each console subscribes to one or more specific channels of the FIT monitor to receive the messages of its interest.

Server side (FIT monitor): An event broker has as primary job to manage all received messages from different probes and subsequently propagate them to consoles in a trustworthy way. Before sending messages each probe has to create an event channel within the event broker. **Intrusion tolerance:** FIT monitor acts as a broker all event messages send by probes. We replicate the broker for providing a trustworthy and fault tolerant monitoring service. To incorporate Byzantine fault tolerance (BFT) into the FIT monitor, we propose to use state machine replication which allows replicas to perform any operation, provided they are deterministic. A brief state of art on BFT protocols and the challenges that still need to be addressed can be found in [2].

3.2 Broker internals

A broker manages all event messages arrived from probes. Figure 2 shows the internal architecture of a broker. Its components are interface, forwarding protocol, ordering protocol and topic-based filter. The interface provides basic methods to create an event channel between publisher and broker. Consoles may subscribe themselves into each channel. When using an interface the user will specify the channel properties such as order, urgency and fault-tolerance. These properties are fixed for each channel based on the application QoS requirements of each event message type.

As can be seen, the broker supports both crash (path II) and Byzantine failure (path I) models. In the case of crash failure model all messages are just forwarded. For the Byzantine failure model, we use SMaRt [1], which implements a BFT protocol. Some events need immediate action, such

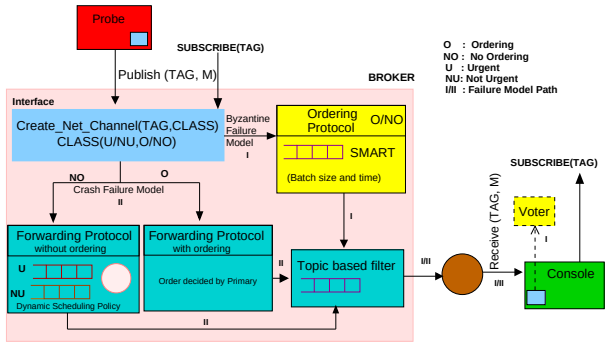

Figure 2: Broker internals

as alarms to the administrator. Other events need to be delivered in total order within a channel or across different channels. This is why the architecture provides QoS parameters for ordering and urgency. They shall be used as needed by user's applications.

4. DISCUSSION

We plan to validate the proposed solution in our cloud infrastructure. We are going to use probes for monitoring IaaS, such as OpenStack and OpenNebula, and consoles to receive and display event messages sent by probes. We are going to simulate crash failures and attacks into the FIT monitoring system intending to verify if it keeps working in a trustworthy and resilient way. The results will be of particular interest to Telecoms which have a special interest in improving the reliability of their monitoring systems.

5. REFERENCES

[1] e. a. Alysson Neves Bessani. BFT-SMaRt - High-performance Byzantine-Fault-Tolerant State Machine Replication, 2011. `http://code.google.com/p/bft-smart/`.

[2] A. Bessani. From byzantine fault tolerance to intrusion tolerance (a position paper). *5th Workshop on Recent Advances in Intrusion-Tolerant Systems (WRAITS), DSN*, 2011.

[3] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Survey*, 35:114–131, June 2003.

[4] R. S. Kazemzadeh and H.-A. Jacobsen. Reliable and highly available distributed publish/subscribe service. In *Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 41–50, 2009.