ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

# Analysis and Comparison between Multiprocessing based and Middle ware based Parallel Remote administration Framework

**Vinay V[1], B.K Srinivas[2]**

Student, Department of Information Science and Engineering, RVCE, Bangalore, India[1]

Assistant Professor, Department of Information Science and Engineering, RVCE, Bangalore, India[2]

**Abstract:** Remote administration is necessary in vast majority of enterprise infrastructure management. Installing software, applying patches or changing system configuration are all part of system administration job. Typically remote administration is done by accessing remote system through network by means of client-server protocol using locally available administration tools (remote terminal or remote desktop. Parallel remote administration frameworks are used in enterprise environment where similar administration tasks need to be performed on many systems. Different approaches are used to achieve remote administration on multiple machines in parallel. One of the method is using multiprocessing and other is using message oriented enterprise middle-ware. This paper gives the analysis of these two different approaches by comparing the performance, scalability, reliability of the framework in enterprise environment. The analysis is done against python 'fabric' api for multiprocessing approach and 'mcollective' orchestration framework for middleware based approach.

**Keywords:** remote administration, agent-based orchestration, fabric api, mcollective

## I. INTRODUCTION

Remote administration can be referred as a method to controlling a computer from a remote location. Software/frameworks that allow the remote administration is becoming increasing common and it is used when it is difficult to be physically near a system to use it[1]. Remote computer may be referring to computer in next room or on the other side of the world. In recent years, remote administration is becoming increasingly necessary where systems are deployed in cloud which is difficult to know the actual location of the physical system.

These machines are administered by accessing the system through cryptographic network protocol SSH (Secure Shell). SSH client is typically used to connect to the SSH daemon running on the remote machine. 'Fabric' is a python library and command-line tool to use SSH for system administration tasks. It provides set of operations for executing local and remote shell command. 'Fabric' api supports parallel execution which is implemented using python multiprocessing module. We can execute set of administration tasks on multiple machines in parallel. So no more running the same tasks machine by machine to make one change on number of machines.

It is simple tool that will make administrator's life so much simpler. Not only can run simple tasks via SSH on multiple machines but you can combine arbitrary python code to make, complex, robust, elegant applications for administration and deployment tasks. "MCollective" is an orchestration framework for parallel job execution systems. "MCollective" uses message broker middleware which supports Publish-Subscribe messaging framework and modern philosophies like real time discovery of network resources using metadata. Used broadcast paradigm for request distribution and all servers get requests at same time. Uses the ability of middlware clustering, routing and network isolation to realise secure and scalable orchestration framework setup. It uses STOMP (Simple Text Oriented Messaging Protocol) to stream the requests and responses so that it is interoperable across different platforms[2].

The goal of this study is to analyse these two different approaches i.e., to compare the performance, scalability and reliability of the framework on large scale system administration. The analysis is configured to orchestrate 16 remote machines in parallel on local LAN.

## II. ARCHITECTURE

### i. *Fabric*

As discussed in introduction part fabric uses SSH to connect to remote machine. In fact Fabric uses paramiko module that implements the SSH2 protocol secure connections to remote machines. Fabric supports both serial and parallel execution of command on remote machine.

Default behaviour is serial i.e., commands will be executed on remote machines one after the other. In order to support parallel execution the Fabric library uses python multiprocessing module library. It creates a number of sub processes equal to number of machines.

Each sub process will make direct connection to remote machine individually using SSH protocol and execute the commands on remote machine. The architecture diagram of python fabric is shown in fig 1.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

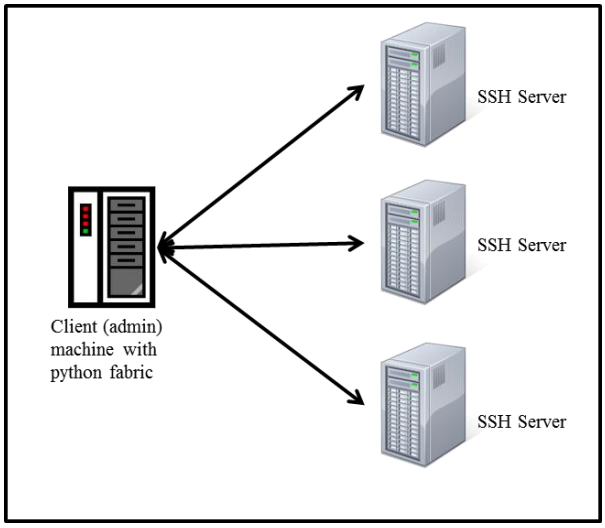



Fig 1. Architecture of python fabric

Python supports following set of commands for fabric api that are simple and powerful.

- **local** - Execute a local command
- **run** – Execute a remote command on all specified hosts with user level permission.
- **sudo** – Execute a remote command on all specified host with sudo permission
- **put** – copy a local file to remote destination
- **get** – download a file from remote server to local system
- **prompt** – prompt the user with text and return the input
- **reboot** – reboot the remote machine, disconnect and wait for wait seconds

*ii. mCollective*

MCollective is a RPC server written in ruby which relies on modern tools like publish – subscribe model which uses enterprise middle to forward request and responses between client and servers. Mcollective communication pattern is highly dependent on routing topology supported by AMQP (Advanced message queuing protocol). Each routing topology of AMQP has its own use cases..

The following are the different scenarios that explain different routing topologies supported by AMQP protocol.

1. Producer – Consumer
2. Worker – Queue
3. Broadcast
4. Message Routing

The conceptual diagram depicting all these routing scenarios are shown in fig 2, fig 3, fig 4 and fig 5 respectively.


Fig 2. Producer - Consumer

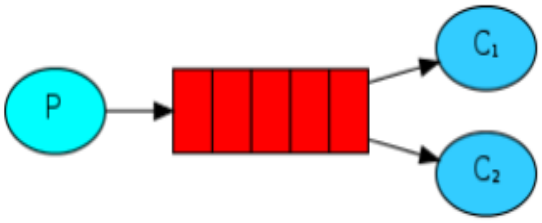



Fig 3. Worker - Queue

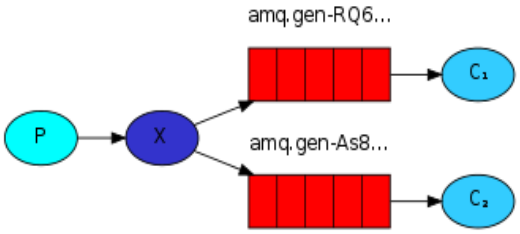



Fig 4. Broadcast

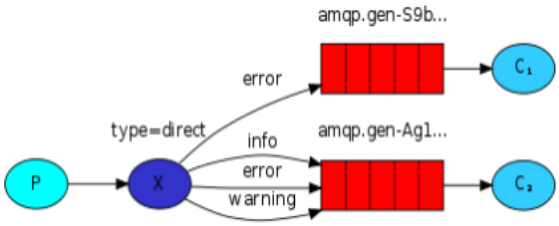
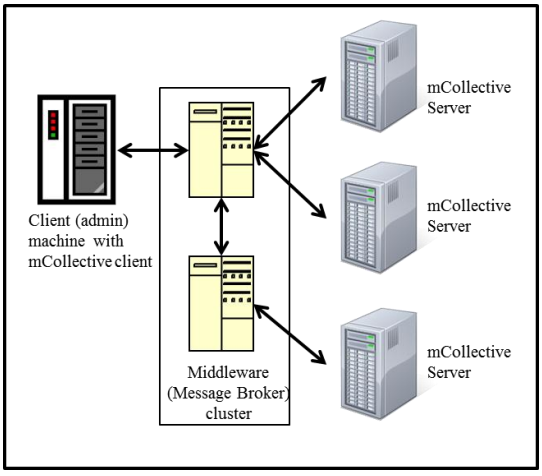



Fig 5. Routing

mCollective uses combination of routing, broadcast and Producer – Consumer routing pattern to communicate with remote server and get the response back from the server to the client.

The architecture diagram of mCollective is shown in fig 6. As shown in architecture the middleware (message-broker) is a dedicated server (rabbitmq or activemq). We can create a cluster of these middleware servers to increase the scalability and reliability of the system. MCollective (server) should be installed on all remote machines and configured to connect to middle-ware with authentication details. similary Mcollective client is installed on different administration machine and configured to connect to middleware with authentication details and metadata information.



ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

Each mCollective server connects to middleware and creates a unique queue in the middleware and subscribes the queue with exchanges by binding queue and exchanges with routing keys. The routing keys that bind queue and exchange will be identified by agent plugins written in each mcollective server. We can write custom plugin to mcollective server based on requirement and platform. When request is initiated from mcollective client, client first discovers the machines for configured timeout period. The mcollective client creates a response queue for each request and sends the request to exchange with appropriate routing key based on the user requests and waits for the response for configured timeout period by listening to response queue, the exchange will deliver the request to the queues by broadcasting the request to the queues that bind to the routing key specified by the client. The mcollective server starts consuming the queue by retrieving each request at a time from the queue. The request will be handled and executed by respective agent plugins in the server. The responses are again queued in response queue created by client. The client starts consuming all the responses from the response queue for configured timeout period. Mcollective client ships with generic rpc client using which we can invoke any agent plugin at mcollective server. This generic rpc client will display the responses from the mcollective on the display in raw format. We can write custom clinet application to customize how to send the request and how to process the responses received by each remote machine[5][6].

## III. COMPARISION

*A.Communication*

In multiprocessing approach the admin machine will make direct connection to each remote machine individually with the help of SSH protocol. This protocol is consumes more network resources and process resource of admin machine which limits the scalability. Whereas in middleware approach there will be no direct connection between admin machine and remote machine. But admin machine will make only one connection with middleware. So in practical it requires same resource whether it needs to administer one remote machine or thousand. So there is not limit for scalability from admin machine point of view.

*B.Scalability*

In multiprocessing approach the numbers of parallel connections that can be made with remote machines are limited by resources of the admin machines. Since number of processes that can be created in a system is limited, number of remote machines that can be administered at a time is also limited. But in middleware approach the middleware is dedicated and can create a cluster of these middleware to increase the scalability of the system. The adding or removing the middleware is dynamic and it doesn't incur a downtime[8].

*C. Reliability*

In multiprocessing approach there is no single point failure, if any one connection terminates, it does not affect the operation of other systems. The reliability depends on the network, if network connection terminates we need to re-initialize the request again for that machine and we cannot resume the previous operation.

In middleware approach if there is only one middleware then the entire operation of the system depends on one middleware. If this middleware fails we cannot communicate with any of the system. But with the ability to create a cluster we can increase reliability of the system. If one machine fails the other machine will still take care of the operation [9].

*D. Security*

In multiprocessing approach we are giving administrator right of the system to the admin. If this administrator is third party we need to compromise the privacy since he has access to the whole system. We cannot restrict the admin to only certain operations.

But in middleware approach the operation he can perform remotely depends on the agent plugins installed on each of the remote machines. And admin will not have access to the entire system. The actual owner of the system can control what third party admin can do to his system.

*E. Interoperability*

Fabric relies on SSH, SSH is platform dependent and doesn't support on all platforms. Mainly it doesn't support world's most widely used operating system "windows". So we cannot administrate windows machine remotely using fabric framework. But mcollective is platform independent, since mcollective is written on platform independent language ruby. The only pre-requisite is ruby interpreter should be available on remote machine. We can write custom plugins for administrator based on platform.

*F. Performance*

Fabric creates separate sub process for each remote machine and creates separate connection attempts for each request and each command it executes. There will be small delay in while creating a process and making a SSH connection with the remote machine. If number of processes increases there will be decrease in system performance due to frequent cpu scheduling and memory swapping.

Whereas mcollective creates only one connection and retains its connection state for ever from remote machine to the middleware. Every request that sends from middleware to remote machine and response from remote machine to middleware uses the same connection. This eliminates the time required to connect and authenticate the system[4].

*G. Additional software requirement*

Fabric uses SSH for connecting to remote machine, so SSH server should be installed and SSH daemon should be running on remote machine. For fabric python interpreter is required with its supporting libraries. For mcollective, mcollective server should be installed and its daemon should be running on remote machine. It also requires some ruby libraries which are used to connect to middleware.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

*International Journal of Advanced Research in Computer and Communication Engineering*
*Vol. 3, Issue 5, May 2014*

## IV. EXPERIMENTAL ANALYSIS

For experimental analysis we used python fabric for multiprocessing based parallel remote administration framework and mcollective for middleware based approach all the remote machines and admin machine are Linux Ubuntu platform. We configured around hundred machines to administrate with both fabric and mcollective. Fig 7 shows the graph that compares the performance between fabric and mcollective.

For experimental purpose we used minimal system requirement setup of admin machine and remote machines are with 128 MB RAM, single core 2.4 GHz Intel Xeon processor Virtual Machine and all machines are in virtual LAN. From the experiment we found that with the above configuration the number of parallel connections from admin machines to remote machines is 18, so we limited the number of parallel connections to 18 at a time.

We tried to get the list of software installed on all remote machines. From the experiment it is found that the time required to administrate increases if number of machines increases in multiprocessing approach whereas in middlware approach the time remains almost constant even if number of machines increases.

## V. CONCLUSION

Remote administration has become one of the most essential requirements in modern day cloud computing era. Since, there are too many different tools are available, enterprise systems admins need very efficient and scalable approach to remote deployment and administration.

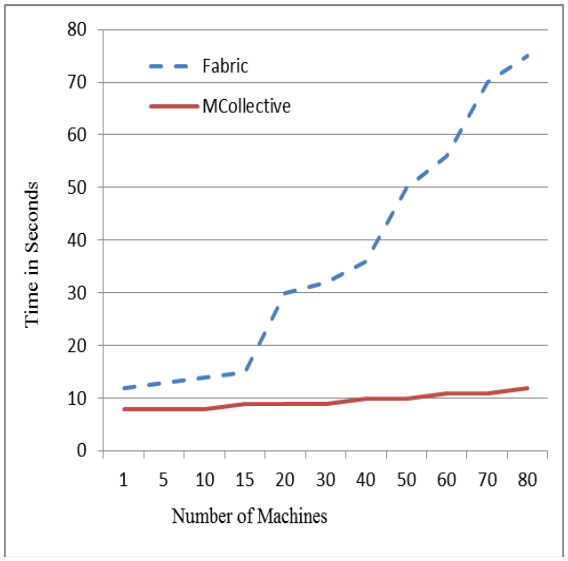



Fig 7. Performance comparison between fabric and mcollective

Above analysis shows the comparison between two different tools which uses two different approaches to orchestrate remote machines. From the experiment analysis it is found that mcollective which uses middleware approach has more efficient, scalable, interoperable and secure compared to fabric which uses multiprocessing approach.

## ACKNOWLEDGEMENT

I would like to thank Mr. B.K Srinivas for guiding me in this work in RVCE college, Bangalore, Karnataka.

## REFERENCES

[1]. Schmelzer, S; von Suchodoletz; Schneider, G; Weingaertener, D; de Bona, L.C.E; Carvalho,C., "Universal remote boot and administration service", Network Operations and Management Symposium (LANOMS), Page 1-6, Oct 2011

[2]. Renhak, K; Seitz, J., "User centric multi-purpose messaging framework" IEEE, International Conference on ICT Convergence (ICTC), Page 66-71, Oct 2013.

[3]. Johnsen, F.T; Bloebaum, T.H; Avlesen, M; Spjelkavik, S; Vik, B., "Evaluation of transport protocols for web services" IEEE, Military Communications and Information systems conference (MCC), Page 1-6, Oct 2013.

[4]. Fernandes, J.L; Lopes, I.C; Rodrigues, J.J.P.C; Ullah, S., "Performance evaluation of Restful Web services and AMQP protocol" IEEE, 5th International Conference on Ubuquitos and Future Networks (ICUFN), page 810-815, July 2013.

[5]. Fan Bai; Wang Tao., "Message Broker Using Asynchronous Method Invocation in Web Service and Its Evaluations" IEEE, 3rd Conference on Software Testing, Verification, and Validation Workshops (ICSTW), Page 265-273, Apr 2010.

[6]. Oh, F.Y.K; Shin-gyu Kim; Hyeonsang Eom; Yeom, H.Y; Jongwon Park; Yongwoo Lee., "A Scalable and adaptive cloud-based message brokering service" IEEE, 13th International Conference on Advanced Communication Technology (ICACT), Page 498-501, Feb 2011.

[7]. Pallickara, S; Gadgil, H; Fox, G., "On the Discovery of Brokers in Distributed Messaging Infrastructures" IEEE, International, Cluster Computing, Page 1-10, Sept 2009.

[8]. Aggarwal, V; Aggarwal, SS; Sharma, V.S; Santharam, A., "A Scalable Master-Worker Architecture" IEEE, SC Companion,High Performance Computing, Netwoking, Storage and Analysis (SCC), Page 1268-1275, Nov 2012.

[9]. Anand M., "Always On: Architecture for High Availability Cloud Applications" IEEE, International Conference on Cloud Computing in Emerging Markets (CCEM), Page 1-5, Oct 2012.