Continuous availability: from the shift paradigm to unmanned operation. Is it still a dream?

M. Capotosto – S. Orsini – P. Tiberi
Banca d'Italia: IT Operations Directorate

Since the beginning, the operational management of traditional data centers has been based on human intervention.
The pervasiveness of the ICT services has brought the data centers to run operations continuously (24/7) and as a consequence there is an exponential increase of the ICT's complexity. As a matter of fact, nowadays, IT infrastructure requires a lot of hands-on maintenance.
Reliance on human operators for managing data centers is the main obstacle for their evolution.
This paper answers the following question: is it really possible to have an Autonomic Data Center?

Introduction

The management of traditional data centers has been based on human intervention. As a matter of fact, IT infrastructure requires a lot of hands-on maintenance and human activities in the day-by-day operations.
Reliance on human operators for managing data centers is the main obstacle for them to move forward, in fact the management of modern data centers is rapidly exceeding human ability, making autonomic approaches essential.

A study was conducted on a new application, having tight constraints in terms of continuous availability (e.g. service continuity, recoverability and resiliency), to determine solutions for autonomic datacenter management.

This paper depicts the operational methodologies used to manage a 24/7 SLA application, applying them on a test environment, representative of the production scenarios.

Four main aspects have been investigated:

1. proactive monitoring;
2. automatic remediation;
3. continuous delivery;
4. dynamic capacity management.

In order to verify all the requirements and the features that should have been satisfied a strategy test was built. This strategy aimed to test the resiliency of our environment and to verify that even with failures / error conditions, it is possible to continue serving our users without interruption and without human intervention.

The results of the study were reported with the willingness to answer the following questions: is really possible to have autonomic datacenter? Is still mandatory to have humans in the Data Center?

Methods

In order to define and verify the operational management methodologies of continuously available applications, a test environment, representative of the production scenarios, was created.

The test plant is constituted by two hypervisors based on two industry standard servers equipped with four 8-core processors and 256 GB of RAM each. The two hypervisors (site A and site B in figure 1), representing two geographically separate data centers, that are interconnected via two 10 Gbit/s network connections. The virtualization software used is VM Ware ESXi 6.0 and inside of each of hypervisor 10 virtual machines, see Figure 1 and Table 1, are deployed:

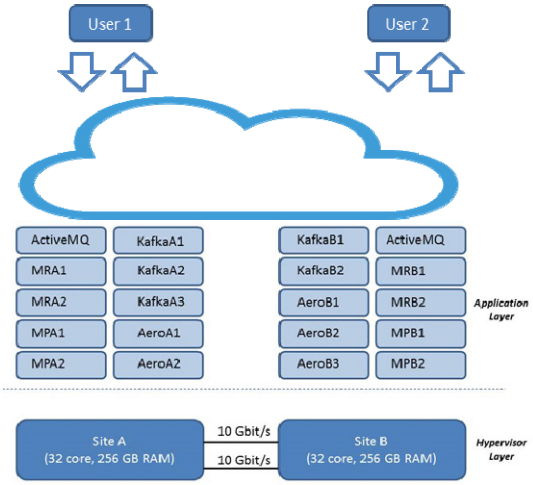

Figure 1 : Test plant architecture schema

Name	*O.S.*	*vCPU*	*RAM*	*Function*
ActiveMQ_A	Centos 7	2	4GB	MQ server
ActiveMQ_B	Centos 7	2	4GB	MQ server
KafkaA1	Centos 7	2	16 GB	Kafka Broker
KafkaA2	Centos 7	2	16 GB	Kafka Broker
KafkaA3	Centos 7	2	16 GB	Kafka Broker
KafkaB1	Centos 7	2	16 GB	Kafka Broker
KafkaB2	Centos 7	2	16 GB	Kafka Broker
AeroA1	Centos 7	2	16 GB	Aerospike node
AeroA2	Centos 7	2	16 GB	Aerospike node
AeroB1	Centos 7	2	16 GB	Aerospike node
AeroB2	Centos 7	2	16 GB	Aerospike node
AeroB3	Centos 7	2	16 GB	Aerospike node
MRA1	Centos 7	4	8 GB	App. Msg. Rout.
MRA2	Centos 7	4	8 GB	App. Msg. Rout.
MRB1	Centos 7	4	8 GB	App. Msg. Rout.
MRB2	Centos 7	4	8 GB	App. Msg. Rout.
MPA1	Centos 7	2	8 GB	App. Msg. Proc.
MPA2	Centos 7	2	8 GB	App. Msg. Proc.
MPB1	Centos 7	2	8 GB	App. Msg. Proc.
MPB2	Centos 7	2	8 GB	App. Msg. Proc.

Table 1 : List of all Virtual Servers configured in the test plant

A test application has been developed in order to simulate a live system. The simple test application is illustrated in Figure 2 and consists of a user part and an application part (the core under test). The user part simulates a typical user behavior and consists of two entities that continuously exchange messages via the core application. Messages are injected/retrieved to/from the core using a messaging front-end exposed by two ActiveMQ servers.

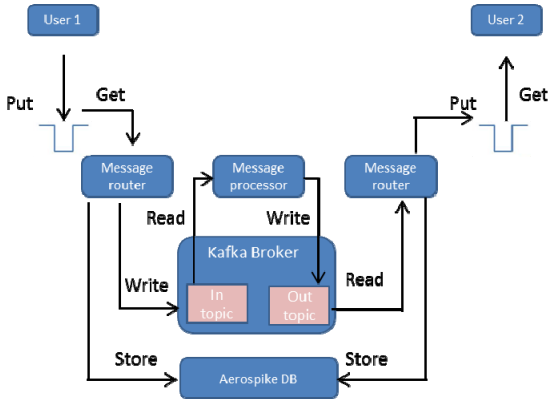

Figure 2: Description of the test application running in the test plant

The core application is composed by:

- Message Router (**MR**): handles communications with Active/MQ servers and writes/reads messages to/from a topic hosted on several Apache Kafka Brokers. The Message Router is also responsible to store each received/sent message into an Aerospike No-SQL database. This part of the application has a redundancy factor = 4 (2 MRs for site A + 2 MRs for site B in active-active configuration);
- Message Processor (**MP**): reads messages from a Kafka Input Topic, performs a simple message manipulation and writes the message to a Kafka Output topic.
  This part of the application has a redundancy factor = 4 (2 MPs for site A + 2 MPs for site B in active-active configuration);
- Kafka Broker (**KB**): manages message communications using a Publisher/Subscriber access method.
  This part of the application has a redundancy factor = 5 (3 KBs for site A + 2 KBs for site B in active-active configuration) with a Kafka replication factor = 3;
- Aerospike Database (**AD**): stores the messages in a No-SQL operational database.
  This part of the application has a redundancy factor = 5 (2 ADs for site A + 3 ADs for site B in active-active configuration) with a Aerospike replication factor = 3.

***

The application under test is open around the clock, without any Maintenance Window (MW) or daily interruption of the service.

How can we achieve the continuous availability?

In order to introduce the continuous availability concept, we have to split it into two main aspects:

- High availability;
- Continuous operation.

High availability is the ability to avoid an unplanned outage. This is achieved with the IT service design (avoid Single Point of Failure, introduce redundancy, detect failures as they occur and so on). As any other IT service, our application may be subject to incidents or failures, which may cause a temporary and unforeseen interruption of the service. However availability of 99.9% should be guaranteed.

Continuous operation refers to the ability to avoid planned outages. The application should continuously operate and mask the planned outage from end-user perspective. This implies that there have to be alternatives to perform necessary administrative tasks (i.e. non-disruptive hardware and software changes, non-disruptive configuration, software coexistence).

Continuous availability combines the characteristics of high availability and continuous operation to provide the ability to keep the application running.

Figure 3 : Continuous availability

Regarding high availability, we have proven experience in designing high availability applications, even if their management is based on reactive monitoring. Whenever the monitoring triggers an event, the on premises personnel (within the working hours; Mon – Fri, 06:30 → 3:00) or the on call personnel (outside the working hours) are in charge of managing and fixing the problem. So, all the ordinary maintenance is based on human intervention.

Therefore, the design of the "old" applications (and of course the business needs) implies the following organizational arrangement:

- shifts: Mon-Fri 06:30 am → 03:00 am;
- working week-ends for maintenance and tests;
- on-call duties.

Regarding the continuous operation aspects, most of the planned maintenance or tests, have been carried out within the MW when the application is closed.

A MW is a period of time during which preventive maintenance that could cause disruption of service, is foreseen.

During MW, nowadays, a lot of activities are carried out (e.g. backups, statistics collection, patch management, operating system package updates, deployment) so human intervention is necessary in such a window; moreover business continuity test (internal and external: supply chain and outsourcer) as well as other important tests are carried out during week-ends when the application is closed.

With applications running continuously, MW concept has become obsolete and all the maintenance activities have to be redesigned.

So to achieve the continuous availability we have to rethink the redundancy architecture of the systems and the applications, overcoming the high availability concept and reconsidering the replication mechanisms in order to make the physical loss of one or more component (or a whole datacenter) completely transparent to the end users.

In particular this application is designed with a distributed shared-nothing architecture that relies on the duplication of processing nodes and the repetitive parallel processing of the same data stream in order to maximize the probability of having always a valid data stream in the face of multiple servers failure or even complete datacenter loss.

Since the application may be subject to incidents, failures and maintenance, and all these events should not impact the application itself, high availability is not sufficient anymore. To get the continuous availability, we have to move to the continuous operations paradigm.

In order to implement the continuous operation four main aspects have been investigated:

1. proactive monitoring;
2. automatic remediation;
3. continuous delivery;
4. dynamic capacity management.

In this context the security issues were not addressed.

1. Proactive monitoring

Proactive monitoring aims to anticipate events before they occur, as well as being prepared and ready for the incident if it were to occur.

Monitoring shall be applied at two different levels: infrastructure (systems and network) and applications to provide a top-down view into the health and status of IT services. Logical and physical security monitoring as well as physical technical plant monitoring are out of scope of this paper.

The infrastructure layer will be in charge of verifying resource utilization such as MHz total of vCPUs, RAM, I/O and system performance such indicators on queues, resource busy or unavailable, network transit time, etc.

Application layer will be in charge of verifying the presence of all modules (keep alive) as well as to continuously check by end-to-end logic if the whole architecture is performing as expected.

A traditional reactive monitoring approach is oriented to trigger on events that can potentially impact an IT

service and to send an alarm to the monitor control room to require human intervention. Moving towards a proactive monitoring approach means addressing problems in advance before they impact the services and possibly remediating automatically (with or without operators on-site).

Monitoring is traditionally based on solicited and unsolicited messages. Solicited messages are produced by a specific command issued by the monitoring tool (e.g. a command to verify the state of a resource such as utilization level of a processor, free space of a file system, memory fragmentation). Solicited messages can be produced on periodic basis (each hour, each quarter etc.). Unsolicited messages are produced by the system autonomously, typically where an error condition arises (e.g. a disk failure, a process abnormal termination). In both cases (solicited and unsolicited) a piece of monitoring logic is activated to analyze the messages and, if it's the case, to submit new commands for collecting additional information; the analysis will trigger a decision and if necessary send an alarm to the control room monitoring system to catch the attention of operators requiring a human intervention. A prioritization color code (green, yellow, red) is used depending on the criticality of the situation and the corresponding reaction time needed. Outside the working hours, an automatic call system is used to make a phone call, with a predefined voice message, to operators associated to that class of problems. The intervention time is immediate during the working hours, whereas a reaction time of maximum one hour is expected outside the working hours.

The proactive monitoring should not wait for the failure to come, but prevent it, as much as possible, by activating some automatic actions. Such analysis cannot be performed by humans when the data center is unattended (e.g. during the week-end). As a consequence the capability of technical monitoring to analyze the situation should be improved to accurately drive an automatic remediation. From a theoretical point of view proactive monitoring should activate a remediation action, carried out by the automatic remediation process, to solve the potential problem. In this way it can seem that proactive monitoring should not change: just invoke an automatic remediation rather than a request for human intervention. This is not the case because the pre-analysis in charge of proactive monitoring should not wait for the event to come (reactive monitoring), but should prevent the event itself (proactive monitoring); moreover the monitoring should not rely, on humans problem determination. Operators, in fact, usually complete the investigation phase before acting on the system. In this sense the bigger difficulty will not be the automatic remediation that will be in charge of a defined action, set of commands, etc. (and can be easily implemented and tested) but the execution of a robust automatic analysis and the consequent choice of one of the automatic actions already built.

Furthermore, once such logic has been developed, tested and successfully implemented there is no reason to not use it also when people are on-site (that means abandon the reactive logic).

It goes without saying that the approach will be incremental and progressive starting from the most frequent and easy cases to cover a high number of problems. Nevertheless not all cases can be tested and operators are still required (especially in the initial phase); technical and organizational measure will be studied to make such intervention effective: e.g. operator will be initially remotely connected to verify how the system is performing.

The implementation of application level monitoring could be realized by building hooks or API endpoints in the application. Planning for application level monitoring in production is one area where DevOps can provide inputs. Messages are setup to notify if the application is not fully or partially accessible or if there are performance issues.

'Monitoring the Monitoring' is another important issue to make sure that the monitoring infrastructure itself is up and running. While disabling such alerting during planned housekeeping must be possible to avoid false positives. If there are multiple instances of the monitoring application running, cross checks can be implemented to verify the availability of hosts used for monitoring.

Some gaps to be covered are:

1. monitor all new components (ActiveMQ, JBoss, Kafka, Aerospike and Zookeeper);
2. automatic opening/closing of trouble tickets on event basis without human intervention;
3. monitoring servers should be duplicated and re-configured to operate without maintenance interruption (capability to operate in 24/7/365, with high availability);
4. automate the monitoring of new nodes in case of automatic scaling of the infrastructure;
5. automatically set a node in maintenance mode when a change activity sets it as "not operational" to avoid false positives;
6. adapt automatically the monitoring rules to the different application versions – if needed – when a change activity sets one node as running in "compatibility" or in "new function" mode;

The work to be done will follow two directions: the technical and organizational part. The technical part will be based on specific tools; the major effort is improving the depth of the analysis to choose the correct action within a list; the organizational part requires a strong integration between proactive monitoring and continuous availability: the two teams should agree on the communication protocol as well as the main events to be processed; furthermore a catalog of predefined

remediation actions should be progressively populated to facilitate the selection of the proper items; a decision algorithm should be implemented in the proactive monitoring to select the action that best fits the situation. Of course the PoC will be based on a few events and a few actions.

2. Automatic remediation

The automatic remediation is aimed to implement automatic workflows that will perform actions to sort out problems. These problems are triggered by the proactive monitoring function.

Traditionally, whenever an incident or failure popped up, the reactive monitoring notified an alert and the on shift technical team took charge of the issue.

Since the project does not foresee any human intervention, we have to put in place some sort of automation in order to react in case of need.

In other words the automatic remediation comes in when the proactive monitoring finds something that needs to be sorted out. This is done with predefined workflows that perform several actions.

To implement the automatic remediation, some tools have been analyzed. The cloud management platform VMware vRealize suite (Operation, Automation and Orchestrator) has been used, although it may be subject to some other solution for the live environment. This choice fits the PoC infrastructure and gives the opportunity to verify the concept. Automatic remediation will offer a catalog of actions to be invocated by proactive monitoring.

3. Continuous delivery

Continuous delivery aims to ensure that the software can be reliably released at any time.

Over the years getting software released has been a risky and time consuming process. The deployment process has been based on the MW concept, so any deployment should have been carried out within dedicated MW in which the service is closed according to predefined SLA.

Each project relies on a custom environment, ideally identical to the production configuration that takes into account the number of instances of the change activity and the specific configuration of the project. After an adequate testing period, the changes are staged in the production environment. Recovery procedures are ready to activate a roll-back in case unexpected problems arise during service time.

Deployment activities are activated manually and can include automatic steps that perform some activity as sub-steps. Automatic steps are implemented to minimize deployment time and to avoid human error because one of the goals of the deploy process is to complete the task in a short time due to the limited duration of the MW.

As tasks are run in the MW, all deployment steps are organized considering that the service is closed. Major changes that can require a long duration, more than the daily MW can typically be delivered during the week-end.

Managing services with 24/7 availability (that means no MW and no week-end) has huge consequences on the organization as well as technical constraints that need to be removed. Automation is still a major driver to reduce to the maximum extent human errors and duration but is not enough.

It's obvious the supposition that in 24/7 services, high reliability is ensured by the fact that there is always a number of available nodes (an even number) therefore a deployment policy that applies the changes in parallel (to minimize execution time) on all nodes in the cluster is disruptive in terms of service availability and cannot be applied.

In a 24/7 service scenario, the deployment process must take into account the underlying architecture of high reliability and act consistently with it. A common solution is deploying on a single node at a time.

This goal can only be achieved by requiring applications that can handle at least two different versions of application software at the same time (the current version (n) and the new version (n + 1)).

By leveraging this requirement, the deployment process must be implemented following a sequential release logic (one node at a time) and should provide for control steps enabling the upgrade to the next node only after checking the proper operation of the current node.

It is therefore clear that rolling back policies must be designed in a way to comply with this deployment scenario. The adoption of a Continuous Delivery-oriented release methodology is therefore necessary. Continuous Delivery (CD) concepts were used to get the deployment process lean and agile.

The goal of CD is to enable a constant flow of changes into production via an automated software production line. The Continuous Delivery pipeline is what makes it all happen.

The pipeline breaks down the software delivery process into stages. Each stage is aimed at verifying the quality of new features from a different angle to validate the new functionality and prevent errors. The pipeline should provide feedback to the team and visibility into the flow of changes to everyone involved in delivering the new feature/s.

A typical CD pipeline will include the following stages: build automation and continuous integration; test automation; deployment/rollback automation.

- Build automation and Continuous Integration.

The pipeline starts by building the binaries to create the deliverables that will be passed to the subsequent stages. New features implemented by the developers are integrated into the central code base on a continuous basis, built and unit tested. This is the most direct feedback cycle that informs the development team about the health of their application code. In this stage the rollback procedure is also tested in order to assure a safe back up environment in case of need.

- Test Automation.

Throughout this stage, the new version of an application is rigorously tested to ensure that it meets all desired system qualities. It is important that all relevant aspects — whether functionality, security, performance or compliance — are verified by the pipeline.

- Deployment/rollback Automation.

A deployment is required every time the application is installed in an environment for testing, but the most critical moment for deployment automation is rollout time. The deployment can be staged, with the new version being initially released to a subset of the production environment and monitored before being completely rolled out. The deployment is automated, allowing for the reliable delivery of new functionality to users within minutes, if needed. Nevertheless if some incompatibilities between version n and n+1 will materialize, an effective and well proven rollback procedure will be carried out.

This process brings the following advantages:

- increases the efficiency and the speed in preparing and managing changes for Production Environment;
- increases developers performance reducing the number of errors and bugs;
- bugs are quickly detected due to the tests being more frequent and more complete;
- application updates are quicker and more secure.

The approach allows developers to transparently implement new retro-compatible features.

In case of non-retro-compatible feature, the application needs to be stopped, causing a planned downtime.

In the perspective of CD, the deployment process must also be automated to avoid manual intervention, to arrange pre-installation and post-installation tasks, to schedule the software build and ready-release providing for an application rollback.

An orchestrator is used in order to reduce manual activities and automate the deployment.

4. Dynamic Capacity management

Capacity Management has to provide IT capacity coinciding with both current and future needs (balanced against justifiable costs). A capacity plan can be created by using a CMIS in order to administer the current usage of service and components and to plan the capacity of the infrastructure to meet the growing or declining needs of the service.

Dynamic capacity optimization is practice of using automation to make adjustments to the resources as needed to accommodate changing workload needs.

The organizational aspects we have currently in place for the IT services in production do not have the need of dynamic capacity management. The analysis is performed in the project phase in order to get the application right-sized to meet the business requirements (capacity planning), then data are collected, reports are produced and on the basis of them actions could be taken. The on-going operation for the capacity management follows:

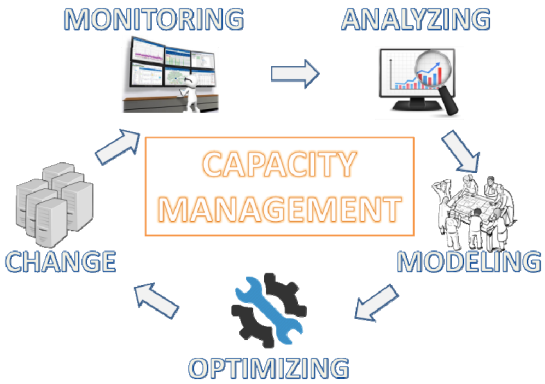

Figure 4 : Capacity Management (as is)

The Capacity Management will be implemented to cope with the new constraints as an automatic process that analyzes all the data, determines trends, what if scenarios and so on to react with automatic workflows. The dynamic Capacity Management process has to:

- predict future requirements and trends;
- plan and implement upgrades;
- seek way to improve and optimize service performance.

The Dynamic Capacity management process has long term visibility and processes the historical data for trend analysis, what-if scenarios and so on.

The Dynamic Capacity Management process has to collect and process the data, and apply predictive analysis to:

- produce an infrastructure consumption trend analysis;
- optimize the resources utilization
- run what-if scenarios

The trend analysis should not be done ex-post by the personnel but in real time and automatically in order to trigger predefined scenarios. This can be done for a more precise analysis (using the analytics) to reduce decision errors (coming from a wrong analysis) and after all to reduce the human effort. Some tools have been analyzed and the cloud management platform VMware vRealize suite (Operation, Automation and Orchestrator) has been selected, although it may be subject to some other solution for the live environment. This choice fits the PoC infrastructure and gives the opportunity to verify the concept.

Moreover this kind of automation helps to get the

seamless operation without human analysis. The data aggregation tracks data center utilization rates and keeps it for IT show back.

***

Test Strategy

All the above features and requirements regarding the operational aspects were analyzed in order to choose and integrate the tools that satisfy them. In this context it has been evaluated that ideally all the requirements should be covered automatically and if some gaps arise, these should be documented. In order to be able to verify the maturity of the instruments, organization and tools to support datacenter management processes for 24/7 applications, a test strategy must be defined that adheres to this context and is able to highlight the key aspects of the various processes. It is therefore necessary to carry out simulation tasks which, by leveraging the reference application of Figure 2, reproduce some operational scenarios on which initiate specific tests of Proactive Monitoring, Automatic Remediation, Continuous Delivery and dynamic Capacity Management processes.

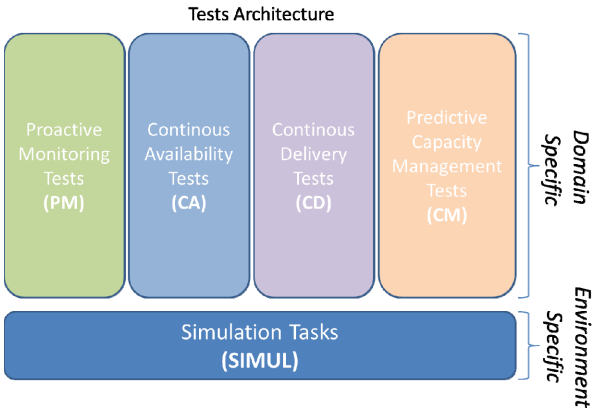

Figure 5 : Test Architecture

Therefore, the test architecture (Fig.5) provides a common simulation layer (SIMUL) on which the various process specific test cases are based.

Simulation tasks are grouped into the following two categories:

a) Normal application behavior & Housekeeping activities. This group includes scripts that simulate the normal user activity by injecting a configurable number of messages with a predefined length at a configurable rate (messages per second). Detailed statistics about number of properly processed messages and total transit time (end to end latency stemming from figure 2: from User 1 to User 2) will be collected for each test; the duration of each test, configured easily by parameter, will be long enough to reach a stable condition, the first period will be excluded from the measurement analysis . From now on we will refer to this group of script as SIMUL.APP.id where id is a numeric indicator to select a specific script within this group. This group also includes scripts that simulate typical housekeeping activities such as Kafka Broker rolling stop/start needed to apply software changes. We will refer to this group of script as SIMUL.HOUSEKEEPING.id where id is a numeric indicator to select a specific script within this group;

b) Error condition. This group contains scripts that simulate anomalous conditions on one or on a set of computing nodes (e.g. Kafka Brokers, Aerospike nodes). Some scripts labelled SIMUL.CPU.id create a specific CPU workload by executing demanding CPU bounded processes like Fast Fourier Transform (FFT) calculations or random number generation involving one or more vCPUs of the computing node under test. SIMUL.MEMORY.id scripts will allocate in exclusive mode a configurable amount of user space memory in order to simulate memory shortage conditions on the node under test.

Process specific test activities will be executed on top of one or more SIMUL scripts and the relevant statistics analyzed in order to verify the effectiveness of the Proactive Monitoring, Automatic Remediation, Continuous Delivery and dynamic Capacity Management actions.

Results

SIMUL.APP.01 script generates an application message with a timestamp and a constant payload of 5120 bytes and sends 1,200,000 messages into an input MQ queue with a constant rate of 1000 messages per second. Each message is processed by the application and a new message is written into an output MQ queue. This output message contains a new field called “latency” that is calculated by the application, using the input timestamp, and taking into account the time needed for the message to pass from the input to the output queue. SIMUL.APP.02 is the same as SIMUL.APP.01 but with an increased message rate of 2000 msg/s which is the upper performance boundary of the application under test while 1000 msg/s is the nominal operating value.

To verify the effectiveness of the delivery process we simulated an infrastructural change applied on all messaging brokers (Kafka Broker) using TEST.CD.INFRA.01 script. This script performs the following actions on all Kafka servers:

1. Stop Kafka server
2. Apply a patch to Kafka server code
3. Start Kafka server

These actions are applied in sequence on each node during a 1 minute interval and all 5 nodes are updated starting from node labelled kafka00 and ending with node kafka04. The script waits for 1 minute before updating the next node.

Figure 6 shows the result of SIMUL.APP.01+ TEST.CD.INFRA.01: all injected messages have been processed by the application (no message lost) and the average latency of 93 ms and a maximum latency of 13.661 ms.

In Figure 6 we reported the 1 minute average of the message latency and we clearly see the spikes corresponding the actions on the different kafka brokers. Just an example we can see a spike around 10:02 that corresponds to the stop of kafka01 node.

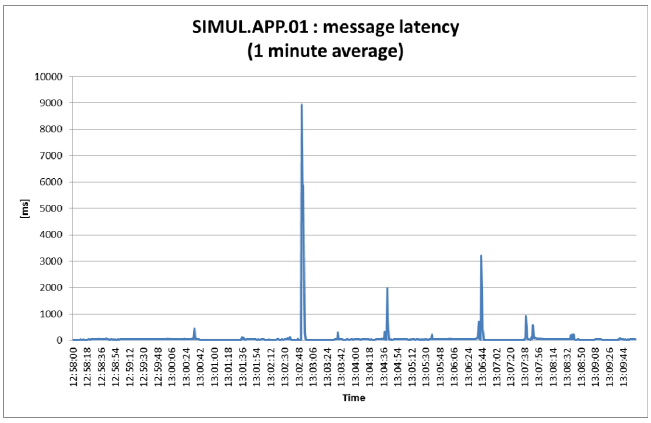

Figure 6 : SIMUL.APP.01

The amplitude and the duration of the spikes can be explained by taking into account the inner properties of the Kafka clusters and its interaction with the clients. We defined one Input topic (Figure 2) with one single partition (i.e. one leader where the clients connect) with replication factor = 3 (i.e. 2 additional internal replicas managed by two Kafka brokers called followers). The Output topic is configured with 10 partitions (i.e. 10 leaders) and with the same replication factor = 3; this means that at any time we have 11 (1 input + 10 output) leaders where the clients are connected and 22 (2 input +20 output) of internal Kafka replicas ready to serve the clients when a leader leaves the cluster.

When we force the closure of a Kafka node, all clients that are connecting to this sever, due to it is a partition leader, faces a timeout and starts a retry routine. The routine ends when one of the followers is elected as the new leader of the affected partition and so clients resume in normal state. This behavior causes a spike in the end to end latency.

The result retrieved from SIMUL.APP.02+TEST.CD.INFRA.01 are reported in Figure 7 and represent a high stress scenario for the application because SIMUL.APP.02 inject a number of messages per second (2000 msg/s) that is the upper processing limit of the application under test.

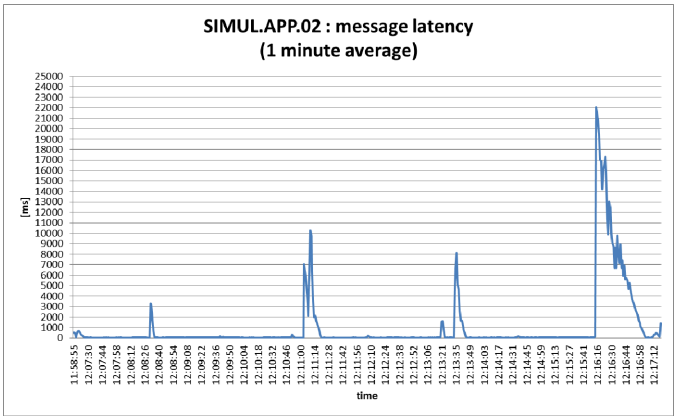

Figure 7 : SIMUL.APP.02

In order to evaluate the effectiveness of proactive monitoring we used SIMUL.APP.02 and on top of it we executed SIMUL.CPU.01 script that added a significant CPU load on all MPA1, MPA2, MPB1 and MPB2 nodes (see Table 1) that host the message processor used to read messages from the input topic and write messages to the output topic (see Figure 2). These nodes, according to Table 1, are configured with 2 vCPUs each and SIMUL.CPU.01 uses a tool to generate a real 100% CPU workload using only two vCPUs. The artificial 100% CPU workload is added in two distinct intervals: the first interval starts at T0 and ends at T1 (i.e. less than 2 minutes) while the second interval starts at T3 and lasts until the end of the test (see figure 8).

TEST.PM.INFRA.01 is used to verify if the monitoring tool is able to detect the high CPU consumption and send a notification to the Automatic Remediation function in order to trigger an action to increase the number of vCPUs available to the message processor nodes.

In figure 8 we can see the end to end message latency measured during TEST.PM.INFRA.01 execution. We can see that the normal average latency is about 45 ms while during interval T0-T1 and T2-T3 the average latency increased to 55 ms as a direct result of high CPU consumption imposed by SIMUL.CPU.01. The proactive monitoring is configured to react to high CPU consumption that lasts for more than 2 minutes so in this case we can see that during interval T0-T1 no action is performed. While during the second high CPU usage interval at time T3 the proactive monitoring detects that the abnormal condition lasts more than two minutes and reacts triggering an increase CPU action via the automatic remediation function.

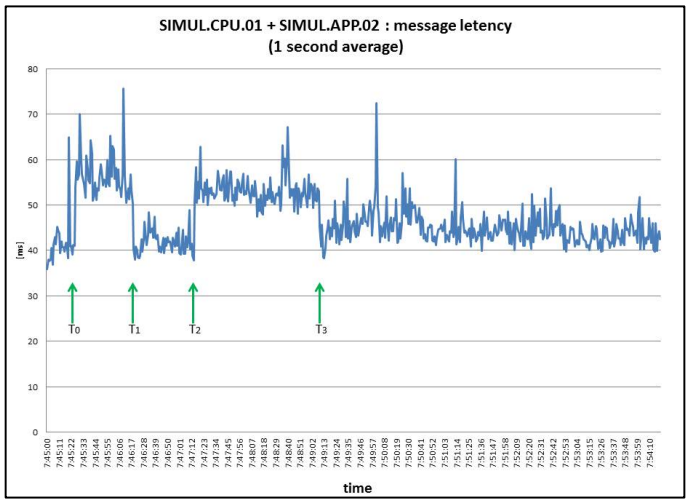

Figure 8 : SIMUL.CPU.01

As a result of this action 2 additional vCPU are hot added to all message processor nodes and the end to end latency decreased to the normal value of 45 ms.

Discussion

As can be observed from Figure 7 the shape of the curve is the same as the one of figure 6 but the magnitude of the impact of TEST.CD.INFRA.01 is amplified in terms of absolute maximum delay and duration of the perturbation. Also in this second case no message has been lost during the test.

If we set a target service level agreement (SLA) of 10 seconds for the latency we can say that while in figure 6 this SLA is always respected in figure 7 the SLA is breached in, at least, two time intervals.

This test can lead us to the conclusion that the continuous delivery process described in Continuous Delivery paragraph is suitable to be used in this scenario. There is a need to carefully plan the time and the relevant workload when changes are applied because they have an effect on the application latency. A possible approach would be to apply changes during low workload hours.

The results of the TEST.PM.INFRA.01 reported in figure 8 demonstrates that the combined action of Proactive Monitoring and Automatic Remediation can be very effective in addressing unexpected CPU workloads and to quickly restore nominal application performance without any manual intervention.

Limitation of the study

Although there is much that remains to be done, our work generates important findings in the unmanned operation field. We can confirm that there are some limitations of this study, caused both by the low amount of tests and scenarios reported and the aspects not yet covered in this work.

Coming back to the scenarios, additional tests have been designed. The preliminary results are still under analysis and not yet formalized.

The tests under discussion are:

Scenario 1: Monitoring system finds that a file system is about to run out of space as the amount of free space is decreasing very quickly.

Action: Monitoring system triggers a file system automatic resize in order to fit the growth rate.

Scenario 2: Monitoring system detects a problem with the integrity check application. It doesn't work or a failure occurs.

Action: Monitoring shut down or isolate the system and on call personnel are notified.

Scenario 3: Site failure

Action: In case of main site failure (Site1 has 3 Kafka brokers and Site2 has just 2) the secondary site has to take over, but needs 3 brokers. The monitoring should catch the site failure and trigger an automatic workflow to start the third Broker (a similar scenario is to be foreseen for the Apache Zookeeper)

Scenario 4: Capacity Analytics (filled in with fake data) finds a trend causing a workload peak the very following day.

Action Capacity system triggers an automatic workflow to create a new node only for the needed period.

Scenario 5: Capacity Analytics finds a constant business growth (or change generally speaking).

Action: Capacity system triggers an automatic workflow to cope with the new business needs.

Conclusion

We have presented a study with the aim to demonstrate the feasibility of new operational management methodologies chosen to manage a 24/7 application. Our results confirm that the methodologies chosen can be used and can form the basis of effective autonomic management.

Although the application is not yet live, our testbed has shown promising results that the application itself can be managed with this operational approach and can run autonomously. In fact no human intervention was required and the application reacted, behaving as expected, against all the failures and generally speaking against all the events proposed. The proactive monitoring detected all the events and triggered the automatic remediation process to run the correct workflow. The continuous deployment approach ensured that the software can be released at any time with the application up and running the dynamic capacity management process exploits data analytics to predict future trends.

As stated before, this is not the live environment, and even though we found pretty good results, only the live production environment can give the actual verdict.

Moreover in order to benefit from this approach, all the applications should run autonomously.

Since the old project, designed with old paradigm (MW, on shift personnel and so on) are hard to change, a phased approach should be taken.

This means that a bimodal data center is a way forward in order to verify new trends and where there is high confidence about the new technologies extend it to new projects.

Finally, under the boundary conditions we talked about, our conclusion is: unmanned is possible if, and only if, we start to think unmanned.

References

[PHIL2014] Andre Philips “The IT Manager’s Guide to Continuous Delivery”
[SIRB2016] Alina Sirbu and Ozalp Babaoglu “Towards Operator-less Data Centers Through Data-Driven, Predictive, Proactive Autonomics”
[POLO2003] Alexandre Polozoff “Proactive Application Monitoring” (2003)

Acknowledgements: The authors want to thank Fabio Fillo for his assistance in the paper review; Cristina Andriani, Alessandro Calvo, Antonino D’Amico, Giuseppe Giangregorio, Sara Lugli, Gianluca Massari, Antonio Possemato, Cristian Sileo for their support.