Impact of Pacemaker failover configuration on mean time to recovery for small cloud clusters

Konstantin Benz, *Researcher, Zürich University of Applied Science,*
and Thomas Michael Bohnert, *Associate Professor, Zürich University of Applied Science*

***Abstract*—In cloud environments High Availability characteristics are established by the usage of failover software (like e. g. HAProxy, Keepalive or Pacemaker). Though these tools enable automatic recovery of cloud services from outages, the recovery can still be very slow if it is not configured adequately. In this paper we developed a "Recovery Time Test" to determine if recovery time depends on configuration of the failover software and how recovery time depends on configuration settings. Another goal of the Recovery Time Test is to determine the factor by which recovery time can be decreased by a given configuration. As proof of concept, we applied the Recovery Time Test to an OpenStack cloud environment which is controlled by the Pacemaker failover software. Pacemaker mean recovery time can take a value between 110 and 160 seconds, if the tool is configured badly. We found that with a proper configuration Pacemaker mean recovery time can be reduced significantly to a value between 15 and 20 seconds.**

Keywords*—High Availability, availability, recovery, fast recovery, failover, Pacemaker, OpenStack, cloud, Recovery Time Test.

I. Introduction

High Availability (HA) can be seen as the "holy grail" in the world of cloud computing [1]. Cloud computing systems should be available without interruptions. End users expect services to be available 24 hours at 7 days per week [2]. Users do not care about how cloud providers can maintain their systems without taking them offline. On the other hand cloud providers want to be able to maintain their services regularly. High Availability tools provide a solution to this problem: some parts of the cloud can be shut down without having to stop the service provided to the end user. Another advantage of HA tools is their capability to recover IT services which were interrupted or stopped unexpectedly. HA tools are capable of running failover actions in case of unplanned IT service outages. Failover software (like HAProxy, Keepalived or Pacemaker) is popular in the cloud computing community. Despite its popularity such software is often unfeasible to recover failed cloud platforms due to the complexity inherent in such systems.

A. Pacemaker as cluster management technology

High Availability software does not run on a single machine. Usually it runs in a cluster: a group of two or more interconnected computers usually referred to as nodes [3]. There are three types of computer clusters: High Availability (HA) clusters, Load Balancing (LB) clusters and High Performance Computer (HPC) clusters [3]. In a HA cluster IT services running on each node are monitored. In the event of failure services are moved from failing nodes to nodes that are still working normally [3]. They remain there until the HA software has recovered the initial state in the failed nodes. In LB clusters some nodes act as a front-end and distribute tasks to back-end nodes depending on the workload [3]. HPC clusters provide the a task distribution to several nodes in order to perform computationally intensive IT services [3]. The distribution of tasks is common to all cluster types and introduces complexity in the overall system architecture. Pacemaker is a typical HA cluster: it monitors different resources on nodes and recovers nodes from outages in the event of failure [3]. Pacemaker has been chosen as the technology to be investigated in this paper because it can handle a HA cluster in a quite generic way: it abstracts from the underlying system environment (number and role of nodes, task distribution rules) by using different resource handlers [3]. Insights about the Pacemaker recovery can be applied to other failover software as well.

Pacemaker is a resource management software which is able to monitor execution of IT services and perform failover tasks if an IT service fails [4]. Pacemaker can manage IT services by using "local resource management demons" (LRMDs) which are shell scripts that are able to stop or restart a single IT service [5]. LRMDs are configured by "resource agents" which define how Pacemaker LRMDs can monitor, start or stop execution of the IT service it observes [5].

B. OpenStack as cloud technology

Failures are not uncommon in cloud environments, although High Availability is an important design feature in almost every cloud architecture [6]. We want to test Pacemaker's ability to recover IT services in the field of small cloud clusters, because Pacemaker is a tool that is supposed to handle failures. Tools like Pacemaker could therefore be an integral part of any cloud computing architecture.

Many cloud platforms are not a monolithic software: OpenStack for example consists in 7 core components [7]. The OpenStack components can be further divided in IT services which must be up and running in order to run the component. Therefore cloud platforms can be considered as groups of IT services. Pacemaker can keep cloud platforms alive only if it can keep all IT services in execution that constitute the platform. If the cloud platform fails, Pacemaker must detect

and recover the failure in all constituting services that stopped execution (and might have caused the cloud platform outage). Therefore Pacemaker needs many dedicated resource agents in order to be able to recover a whole cloud software.
OpenStack is currently evolving at a rapid pace and in short development cycles [8]. It has a fast growing support community compared to other open-source cloud platforms like OpenNebula [9]. OpenStack is fully compatible to popular commercial cloud platforms like Amazon EC2 [12]. Therefore OpenStack is a feature-rich cloud platform which is similar to many other cloud technologies [12]. We want to test Pacemaker in an OpenStack environment, because observations on how Pacemaker handles OpenStack can be applied to other cloud platforms as well.

C. Goals of the evaluation

Though Pacemaker can handle many different types of IT services, the time to recover a whole cloud platform from an outage depends on how fast Pacemaker can recover the individual IT services that constitute the cloud platform. The management of IT service recovery tasks is performed by the Pacemaker "Cluster Resource Manager" (CRM) component. The CRM configuration is contained in an XML file, the "Cluster Information Base" (CIB). The CIB can influence the order as well as the colocation of IT service recovery tasks.
One could easily imagine that the order of recovery tasks could influence the recovery time of a cloud platform, since cloud platforms are not monolithic. Therefore we want to test if and how different CIB configurations are able to speed up the recovery of a cloud platform.
For this purpose we developed the "Recovery Time Test" procedure. In this test we simulate IT service outages in a cloud platform which is managed by Pacemaker. Then we measure the recovery time Pacemaker takes to recover the cloud platform completely from an outage. The test is performed with different CIB configurations. Then the recovery times of test runs with different CIB configurations are compared to each other.
The goal of the Recovery Time Test is to find out:

1) If the CIB configuration has a **significant influence on the recovery time**.
2) Which CIB configurations generate **significantly faster recovery** for OpenStack than others.
3) To estimate **the factor by which the recovery time can be reduced** by optimizing the CIB configuration only.

D. Related Work

Our analysis is limited to an OpenStack implementation which is run on two nodes, because we want to analyse the influence of the CIB configuration on the recovery time. We are neither interested in the influence of the cluster size on the recovery time nor do we compare recovery times of different cloud platforms. Though this may be a rather specific topic for scientific research, we target more general areas in the theory of High Availability design too.
The influence of configuration files on system availability and recovery behavior has not been investigated with statistical methods so far. There are some reports on the benefits of employing simulation tools like the "Chaos Monkey" [10]. Those reports lack statistical evidence of the inherent value in using of such tools in order to design reliable systems. This paper is a first step towards closing this gap. Outage simulation tools could be used to proof validity of well-known principles for the design of High Availability systems like e. g. "Design for Six Sigma" [11]. The current paper investigates simulation of outages in order to create a more systematic approach in designing reliable infrastructures.
A comparison of OpenStack with multiple other open-source cloud platforms can be found in [12]. The authors of this study investigate (among other factors) the scalability of different cloud platforms [12]. One of their conclusions is that a small OpenStack cluster behaves quite differently from a large deployment involving hundreds of servers [12]. An analysis on the Pacemaker recovery behaviour in larger OpenStack clusters has not been performed at the time of this writing, although it will clearly have a significant impact and might be the subject of future research activities.
Other related work concern different HA recovery strategies and how they can be applied to cloud environments [13]. It could be an interesting future research topic to evaluate how Pacemaker can implement those strategies in comparison to different alternative HA technologies. Such an investigation is especially interesting when one wants to use HA technologies in order to optimize power consumption in cloud environments [14].

E. Structure of paper

In the following section (section II) we explain the Pacemaker functionality and how Pacemaker is integrated into the OpenStack architecture we want to investigate. In section III we describe the Recovery Time Test and how the recovery time data is gathered. In section IV we analyze and discuss the results of the Recovery Time Test. This paper concludes with some recommendations on how Pacemaker recovery could be enhanced.

II. PACEMAKER

Pacemaker is a distributed software [4] which uses two main components:

1) **Cluster Resource Manager (CRM):** The CRM is responsible for managing different resources in a cluster of (physical or virtual) computers. [15] A resource could be any kind of IT service running on the cluster nodes: an IP address, a database server or a shell script. The CRM itself does not directly perform resource management tasks like monitoring, stopping or starting IT services. Those tasks are performed by "Local Resource Management Daemons" (LRMDs). [15] LRMDs are IT services which run locally on each cluster node and are able to perform shell scripts - LRMDs must be configured by using "Resource

Agents" which tell them how they can monitor, start or stop an IT service. [15] The LRMDs can only perform tasks which are defined in "resource agents". The CRM can be seen as the coordinator of the LRMD activities. The CRM tells each LRMD when (at what time interval) and where it must perform the resource management tasks. The CRM also defines which resource management actions must (and must not) be performed together.The CRM orchestration is defined by a "Cluster Information Base" (CIB) file. [15]

2) **Cluster Communication Manager (CCM):** The CCM is a distributed application which runs on all cluster nodes and provides communication between CRM and LRMDs. [15] The CCM itself is not a Pacemaker component, but it is a component which is required by Pacemaker in order to work properly. Typical CCMs which are used with Pacemaker are Heartbeat and Corosync. [15] The CCM is configured by local configuration files which must be installed locally on all cluster nodes. All CCM configuration files must be identical on all cluster nodes.

Due to its well-separated architecture Pacemaker is very flexible: it is able to run in heterogenous cluster environments, it is customizable and it can automate failover tasks for many different kinds of IT services. On the other hand the Pacemaker orchestration of failover activities (which is done by the CRM) can become very complex and difficult to configure.

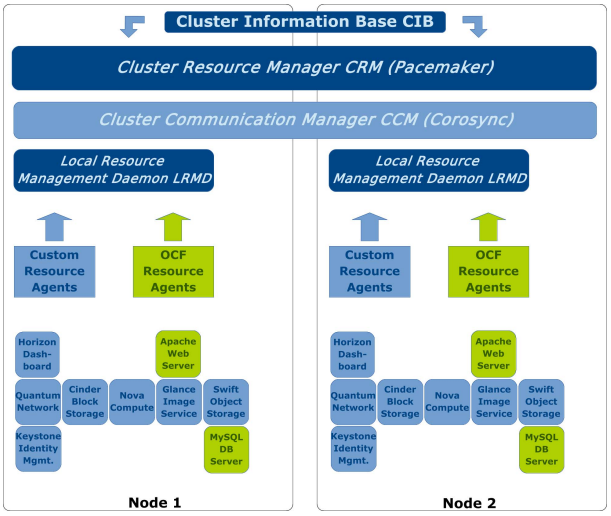

FIG. 1: Integration of Pacemaker in the OpenStack HA cluster.

A. Integration in OpenStack

An OpenStack cluster can be managed by Pacemaker in order to become highly available. There are some tasks required [16] to integrate Pacemaker into OpenStack:

- A standard Pacemaker installation is not delivered with already integrated resource agents for OpenStack services. Therefore custom resource agents must be written for each IT service which is required to run OpenStack. The resource agents must be installed locally on each OpenStack node to allow the LRMDs to perform failover tasks. [17]
- A CCM must be installed and configured on all cluster nodes. For our purpose we use Corosync as CCM. The CCM must run on all cluster nodes before Pacemaker can be started. [18]
- Finally Pacemaker is installed on all cluster nodes. [19] Pacemaker can be configured by editing its CIB file. The CIB file must contain failover tasks for all IT services which are required to operate OpenStack.

Once these tasks are performed Pacemaker is integrated into OpenStack. The architecture of the Pacemaker integration into a two node OpenStack installation can be seen in (Fig. 1). This architectural setup will be used in the Recovery Time Test. The reason for using a 2 node cluster is that we want to test recovery behavior when a single fallback component is employed rather than propagation of recovery mechanisms in larger clusters.

III. RECOVERY TIME TEST SETUP

The idea of the Recovery Time Test is to measure how long it takes until Pacemaker restores an OpenStack outage. It relies on the "Chaos Monkey" test described in the "Dependability Modeling Framework." (DMF). [20] After random outages have been simulated (using a so-called "Chaos Monkey" tool [21]), the recovery behaviour of the system is measured.

The Recovery Time Test must be an automated test, because measuring the recovery time of fast recovering IT services manually would be a very imprecise method. Therefore test runs are controlled by a computer program (in our case a Python script) that runs on a machine which is located **outside** of the OpenStack cluster that is tested. This "supervisor node" should have remote access to internal nodes of the OpenStack cluster in order to be able to run tests and observe system activities on them. The supervisor node is not a cluster node itself, because we do not want that the test script interferes with Pacemaker recovery activities. We only want to be able to start recovery procedures on cluster nodes and observe recovery behaviour, but we do not want to manipulate the recovery behaviour itself.

In our setup, a typical test run is a two node OpenStack installation which forms a Pacemaker cluster. As a first step some OpenStack services are randomly (and remotely) shutdown by the test program. Immediately after having sent out the shutdown signal and having received an acknowledgement that the service has been interrupted, the test script starts a timer. Then Pacemaker should start its automatic recovery procedures on the cluster nodes. Meanwhile the execution status of the OpenStack services is polled repeatedly by the test script in the supervisor node. Once the supervisor node detects that all OpenStack services are recovered, the timer is stopped and the total recovery time is measured and stored in a database on the supervisor node.

The Pacemaker recovery time is not expected to be fully

deterministic: if we perform a test run once and repeat the same test on the same nodes (with the same CIB configuration) again, the obtained result might not be the same. The cluster nodes are computers and therefore they are subject to random errors [22]. In order to deal with this random noise, for each CIB file configuration multiple test runs were performed and the total recovery time was averaged over all test runs. This method of averaging values should cancel out errors produced by random noise [22].
The goal is to find out if a relationship between CIB file configuration and average recovery time exists. Our assumption is that a "smart" distribution of recovery tasks could improve recovery speed in Pacemaker. Therefore test runs were not performed only on a single OpenStack installation. The test runs were performed with multiple different CIB files and then grouped according to the different possible Pacemaker CIB file configurations.

A. Pacemaker configurations

The Pacemaker CRM component is configured mainly by the CIB file. This file tells the CRM **when** and **where** it has to start a particular recovery task. According to this separation a CIB configuration can be optimized in two areas:

1) Intervals and timeout of recovery tasks ("when").
2) Grouping and colocation of recovery tasks ("where").

Even when we use Pacemaker in the limited setup depicted in figure 1, there is still an indefinite number of possible CIB configurations. Since we are not able to test all possible CIB configurations with the Recovery Time Test, we must create a sample which is representative for all other possible CIB configurations. Therefore we decide to test only configurations which represent particular interval time, timeout value, grouping or colocation characteristics.

TABLE I: Discretization and categorization mapping of variables.

Parameter	Selected Value	Mappping
Interval time	5 seconds	Large interval time
	1 second	Small interval time
Timeout	60 seconds	Large timeout value
	30 seconds	Small timeout value
Group	All services contained in one resource group	All services in one group
	All services grouped by the OpenStack component they belong to	Services grouped by function
	All services kept as atomic services	Services not grouped
Colocation	All services run on one node only	Active on one node
	All services run on both nodes in parallel	Active on both nodes

Our sampling strategy is to group the different CIB configurations alongside these four characteristics and all possible combinations of it. The combination of characteristics raises another issue if we consider that interval time and timeout value are not discrete parameters: since both parameters are not discrete, we have an infinite number of possible combinations. Therefore we must discretize interval time and timeout values into a discrete number of categories. For the sake of simplicity we chose to use two values for the recovery task interval time which are seen as belonging to either the category of "large" values or the category of "small" values. This way we can group the CIB configurations into configurations with small or large interval time and discretize the recovery task interval time parameter. The same discretization is applied to the timeout parameter. In table I we show how two particular manifestations of interval time and timeout parameters are mapped into the two target categories.
Another issue is that the possible ways to group recovery tasks depends on the number of services and is possibly very large. In our current architecture we have 24 services which are managed by Pacemaker. There are more than 6×10^{23} groupings of services possible. We choose to simplify the grouping task by introducing three levels of group granularity. We group the services as either atomic services (which are not grouped at all or grouped with maximum granularity) or grouped by their purpose they have in OpenStack (see table I) or as one group containing all OpenStack services (no group granularity at all).
Another issue is that it is unclear how we should define colocation of multiple active services on a single node. Again we introduce different levels of colocation: either there is no colocation of actively running OpenStack services required (all OpenStack services are allowed to actively run on both nodes in parallel) or there is a colocation of services which allows services to run only on one node at once. Table I shows the mapping of colocation manifestations into two categories.
As a result of our discretization and categorization efforts we transform the interval time, timeout, group and colocation variables into categorical variables. Furthermore we can turn the four variables into one single categorical variable by unifying all possible combinations of manifestations of the variables in one single categorical variable: the configuration variable. This variable consists in all possible 24 combinations of interval time, timeout, grouping and colocation.

B. Simulation of outages

The Recovery Time Test is a test which is performed for one of the possible Pacemaker configurations we show in table **??**. We start with the OpenStack architecture in figure 1 and use one of the possible CIB configurations. Then we simulate some service outages and measure how long it takes for Pacemaker to completely restore all OpenStack services.
One test run consists in the following steps:

1) Shut down a number of OpenStack services on one of the nodes and start a timer to measure the recovery time.
2) Let Pacemaker execute failover tasks and check which OpenStack services are available by repeatedly polling their execution status.
3) If all OpenStack services are available again the timer should stop and indicate the recovery time.
4) Store the recovery time in a database and clean up all system modifications performed during the test run.

The services which are shutdown in the first step must be chosen at random because the recovery time must be a random number. The underlying assumption is that service outages generally occur at random and it can not easily determined which service will fail next.
One test run reveals only one possible recovery time value and might have occurred accidentially. A test run is not representative for the general failover behavior of the Pacemaker software for a given CIB configuration. Therefore we must apply multiple test runs for each of the 24 configurations we want to test.
The problem is that running thousands of tests is computationally expensive. The 24 OpenStack services are dependent to each other. If one single service is interrupted, the interruption of this service can lead to failure of other services as well. In the worst case, an interruption of one single service leads to failure of all services. Since we have monitoring intervals of up to 5 seconds, Pacemaker needs $24 \times 5 = 120$ seconds to detect such a full outage of all services. The time consumed by the recovery procedure must be added to this value. If we assume a recovery time of 0.5 seconds per single service, it takes $24 \times 0.5 = 12$ seconds to restart all services. Additionally the recovery procedure is sometimes restarted when the timeout value for recovery of a single service is reached. If we assume a single restart of the recovery procedure in the worst case (timeout = 60 seconds), the recovery procedure can consume 84 seconds. A single test run can consume more than 3 minutes. Running 1'000 tests 24 times will result in a total computation time of more than 50 days, which is not a feasible time horizon from a practical point of view.
On the other hand we must take at least some test runs in order to be able to reason about the data. Therefore we want to take a set of test runs which is small enough to be performed in a reasonable amount of computation time, but sufficiently big enough to draw valid statistical conclusions from the data. We need a sample size which is representative to the number of outages in the real lifetime of a cloud service.
A comprehensive treatise on the lifetime of IT services can be found in [23]. According to the view of the authors, any large piece of software can be expected to have an average lifetime of 12.3 years [23]. This means that a cloud platform can be expected to run for 147.6 months. If we want to estimate the number of outages in the lifetime of a cloud service, it is adviseable to get some empirical data of outages that occur in productive commercial cloud solutions. One such study can be found in [24]. According to this study, an outage of commercial cloud platforms (like Google, Microsoft etc.) can be expected to occur every 3 months [24].
If we presume that such a rate of outages will occur in the lifetime of a cloud platform, we can estimate that there might be about 50 outages in the lifetime of a cloud service. We chose to run the Recovery Time Test 30 times for each configuration in order to retrieve a realistic recovery time value. A sample of this size covers more than 60 % of all outages that might occur in a productive cloud platform and is feasible to many statistical tests.
The choice of 30 test runs may be a simplification, but since much larger samples are impractical to compute, it serves our purposes well enough. It is sufficient to notice that the sample size is not too small to draw conclusions from the data. Our data set of test results will consist in 24 groups with 30 recovery time values per group.

C. Recovery time measurements

For each test run we must measure the time between the outage and the full recovery of all OpenStack services. For these recovery time measurements we use two instruments:

1) As we denoted in subsection III-B we must use a timer which starts when the random service is shut down and which stops when all OpenStack services are available.
2) In order to know when we can stop the timer again we must repeatedly poll the execution status of all OpenStack services. Once all OpenStack services are up again, we must stop the timer and store its value as result of the test run.

While the timer does not significantly influence the recovery time measurements, the polling interval basically determines the precision of measured values. The interval should not be too large because we want to measure differences between recovery times. On the other hand we do not want the poll requests to consume too many resources.
This issue can be solved if we consider that the outage recovery time must be related to the upstart time of an IT service. If e. g. one OpenStack service fails completely and there is no redundant service available, the Pacemaker recovery takes at least as much time as is required to restart the failed OpenStack service. From that point of view it does not make sense to have a polling intervall which is much smaller than the average restart time of an OpenStack service. Therefore we chose the polling interval to be 10 times smaller than the average upstart time of an OpenStack service. The average upstart time is evaluated experimentally by restarting each service 100 times, measuring the upstart time and calculating the average.
The measurement of recovery times takes place by shutting down one service, starting a timer, polling if all OpenStack services are available and stopping the timer if all OpenStack services are available again. The timer value will be stored as result of one test run.
For the 24 different CIB configurations we must also calculate the average and variance of all recovery times measured per CIB configuration. This prerequisite is required to perform statistical analysis with the gathered data because we must know how the recovery time values are distributed for each CIB configuration. The goal of our test is to find and analyze any kind of correlation between CIB configuration and recovery time.

D. Test methods

The first question we must answer with the Recovery Time Test is if there is a significant influence of the CIB configuration on the recovery time. This will be tested with a one-way ANOVA test.
A CIB configuration which is tested by the Recovery Time Test

can be seen as a sample that belongs to a population which is different to the population of another CIB configuration. A CIB configuration with e. g. small interval time belongs to another population than a CIB configuration with large interval time. Since we have different samples belonging to different populations we must compare mean recovery times of all different samples with an adequate statistical instrument like the one-way ANOVA. [25]

This test is applicable when the prerequisites to use a one-way ANOVA are fullfiled. These prerequisites are homogenity of variance and a gaussian distribution of the average recovery time. [25] Gaussian distribution can be verified by applying a Kolmogorov-Smirnov test to all measurements and a normal distribution with equal mean and variance. [26] Homogenity of variance can proven by a Levene test. [27] If one of these tests fails, a Kruskal-Wallis H-Test must be performed instead of a one-way ANOVA. [28]

We prefer one-way statistical tests to their multi-way alternatives because the latter are computationally expensive and can be replaced if we define the CIB configuration as one single categorical variable.

The second question we have to answer is which CIB configurations do improve the recovery speed. This can be done by performing a posthoc test that reveals which of the 24 CIB configurations significantly differ from the others. Since we are interested only in configurations that improve the recovery velocity, we can eliminate all CIB configurations which produce larger, equal or non-significantly lower recovery time. As a result we get configurations which enhance the Pacemaker recovery performance.

Our third interest is the size of the factor by which mean recovery time can be reduced. This size can be evaluated by finding a (minimal) linear regression model for the size of the recovery time in relation to a CIB configuration value [29]. For this purpose we must find a sufficiently good fit for the regression equation 1.

$$z_k = \beta_0 + \sum_{i=1}^{N} (\beta_i \times f_i(x_k)) + u_k \tag{1}$$

z_k is the actual recovery time, β_0 and β_i are the regression coefficients, $f_i(x_k)$ is a (local) function of the value of the CIB configuration, N is the number of independent variables (the four CIB configuration characteristics) and u_k is an error term. We recursively formulate the regression function eliminating all variables which turn out to be non-significant and do not explain recovery time. As a result we get a minimal model which describes relation between recovery time and CIB configuration values. In order to calculate the factor by which recovery time can be decreased, we have to evaluate (allowed) minimum and maximum of the regression function. The factor is the maximum value divided by the minimum value.

IV. Results of Recovery Time Test

A. Dependence of recovery time on Pacemaker configuration

Our statistical tests reveal that there are indeed significant recovery time differences between differently configured OpenStack Pacemaker clusters. Table II shows sample size, mean, standard deviation and standard error of recovery times measured for each configuration.

Table II also shows lower and the upper bounds of the 95%-confidence interval as well as minimum and maximum value for each group. There are some quite obvious differences between configurations where OpenStack services are managed as one single group and configurations where Pacemaker manages all services separately. Ungrouped resources tend to be recovered much faster than resources managed as a group.

In order to test if these differences between mean values did not occur by chance alone, we must first check which statistical test should be applied. Therefore we apply the Kolmogorov-Smirnov test to see if all samples are distributed normally and an ANOVA is applicable.

TABLE II: Different recovery time samples taken for each configuration.

			Std.	Std.	95% - Conf. Int.			
Config.	N	Mean	Dev.	Err.	Low.	Upp.	Min.	Max.
LLG1	30	130.58	83.33	15.21	100.76	160.40	5.83	314.75
LLG2	30	141.95	73.11	13.35	115.79	168.11	30.85	301.20
LLI1	30	37.48	25.58	4.67	28.33	46.63	4.74	106.24
LLI2	30	74.59	53.09	9.69	55.60	93.59	3.33	189.29
LLA1	30	13.63	6.04	1.10	11.47	15.79	3.13	26.52
LLA2	30	16.59	11.01	2.01	12.65	20.53	3.07	53.71
LSG1	30	99.32	62.96	11.50	76.79	121.85	4.66	212.26
LSG2	30	93.85	40.91	7.47	79.21	108.49	6.79	159.37
LSI1	30	37.22	27.96	5.10	27.22	47.23	9.37	116.89
LSI2	30	92.16	52.59	9.60	73.34	110.98	10.98	186.42
LSA1	30	13.61	7.46	1.36	10.94	16.28	3.58	47.13
LSA2	30	111.66	53.16	9.71	92.64	130.68	11.88	186.58
SLG1	30	158.19	82.62	15.08	128.63	187.76	3.33	360.95
SLG2	30	118.98	59.62	10.89	97.64	140.31	3.52	238.86
SLI1	30	46.31	34.08	6.22	34.11	58.50	3.68	151.03
SLI2	30	139.74	82.18	15.00	110.33	169.14	3.34	293.57
SLA1	30	20.49	12.27	2.24	16.10	24.88	3.71	68.09
SLA2	30	33.00	25.60	4.67	23.84	42.16	5.98	113.40
SSG1	30	115.12	74.85	13.67	88.34	141.91	2.86	227.81
SSG2	30	119.43	78.16	14.27	91.46	147.40	3.18	228.36
SSI1	30	60.06	40.14	7.33	45.70	74.43	4.01	169.47
SSI2	30	117.32	71.75	13.10	91.65	143.00	3.54	232.48
SSA1	30	18.83	12.86	2.35	14.23	23.43	3.46	67.28
SSA2	30	128.91	67.34	12.30	104.81	153.01	40.75	273.44

The whole set of recovery time measurements contains 720 measurements. The mean is 80.79 seconds and the standard deviation equals 71.15 seconds. When the set is compared to a normal distribution with $\mu = 80.79$ and $\sigma = 71.15$ applying the Kolmogorov-Smirnov test, we get $D = 0.1847$ and a p-value of 4.278×10^{-11} which means (under the assumption that $\alpha = 0.05$) that the data is not normally distributed.

In the QQ-plot (figure 2) the theoretical quantiles of a normal distribution (with mean=80.79 and standard deviation=71.15) are plotted versus the sample data points obtained in our measurements. If the data were normally distributed, all data points would lie close to the straight line in the middle of the graph. Instead of following the line, the distribution of data points is skewed extremely to the left. In a QQ-plot this is an indicator that the data is not normally distributed [30].

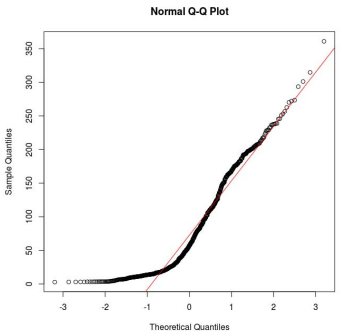

FIG. 2: QQ-Plot of recovery time measurements versus a normal distribution with equal mean and variance.

The QQ-Plot shows that the first quartile is following a much "flatter" distribution than the other quartiles. Configurations with higher recovery time are generally configurations which cause Pacemaker to restart failed services in a particular sequence. Since this complex restart order involves many dependencies between services, we can observe many restart failures resulting in a much "steeper" distribution of "high" recovery times.
A further Levene test for homoscedascity is obsolete, because the conditions for using an ANOVA are violated. Therefore we will use the Kruskal-Wallis H-Test in order to check if different samples (which use a different Pacemaker configuration) have significantly different means.

TABLE III: Results of the Kruskal-Wallis rank sum test.

	Degrees of Freedom	Chi-squared	Probability P(> F)
Configuration	23	343.5559	$< 2.2 \times 10^{-16}$

If we assume that there are no differences between mean recovery times of different samples (the recovery time does not depend on Pacemaker configuration) and test this hypothesis for an $\alpha = 5\%$ using the Kruskal-Wallis H-Test, we must reject the hypothesis that recovery time is independent of the configuration. Table III shows that the p-value of the test is smaller than α. The Pacemaker recovery time depends on the CIB configuration.

B. Pacemaker configurations leading to shorter recovery time

With the results of the Kruskal-Wallis test, we have proven that the CIB configuration influences the recovery time. In order to find out which configurations reduce the recovery time significantly, we have to perform a posthoc-test. We choose the Kruskal-Wallis Multiple Comparison test, which pairwise tests if the difference of means of two groups is significant. [31] Since we have 24 configurations, there can be potentially many groups which have a significant difference in means compared to other groups. Therefore we counted how many significant negative differences (in mean values) to other groups are available per configuration group. The result can be seen in table IV.

TABLE IV: Number of groups which have significantly lower mean recovery time.

Interval		Timeout		Group		Colocation	
5s	54	60s	52	All	0	Yes	52
1s	20	30s	22	By function	14	No	22
				None	60		

Obviously the Pacemaker recovery time can be reduced by using rather large monitoring intervals and larger timeouts. It is adviseable to colocate resources, but the OpenStack services should not be put in a large group. Pacemaker seems to face difficulties when it has to recover many IT services at once. Ungrouping of services leads to faster recovery times.

C. Recovery time decrease potential

In order to determine the recovery time decrease potential we have to perform a linear regression using the monitoring interval, timeout value, grouping type and colocation type as explanatory variables and the recovery time as output variable. Grouping is encoded as an ordinal value starting with 1 as strongest grouping (all OpenStack services in one group) and 3 as weakest grouping (Pacemaker manages all services individually). The colocation type is encoded in an analogous manner. Starting with our first regression model we get the results listed in table V.

TABLE V: Regression coefficient estimate for a recovery time model involving interval, timeout, grouping and colocation variable.

Coefficient	Estimate	Std. Error	t-value	Probability Pr(> $\|t\|$)
(Intercept)	126.57	11.76	10.76	$< 2 \times 10^{-16}$
Interval	-4.45	1.13	-3.95	8.51×10^{-5}
Timeout	-0.21	0.15	-1.40	0.16
Group	-38.79	2.76	-14.06	$< 2 \times 10^{-16}$
Colocation	36.45	4.50	8.09	2.61×10^{-15}

As we can see, the timeout value does not have any significant impact (p-value is smaller than $\alpha = 0.05$) on the recovery time. Therefore we reduce our model and get the results in table VI.

TABLE VI: Regression coefficient estimate for a recovery time model involving interval, grouping and colocation variable.

Coefficient	Estimate	Std. Error	t-value	Probability Pr(> $\|t\|$)
(Intercept)	117.07	9.63	12.15	$< 2 \times 10^{-16}$
Interval	-4.45	1.13	-3.95	8.6×10^{-5}
Group	-38.80	2.76	-14.05	$< 2 \times 10^{-16}$
Colocation	36.45	4.51	8.08	2.71×10^{-15}

The regression equation can be found below (equation 2). When we enter a high interval (5 seconds), weak grouping and high colocation value, we get a theoretical minimum of 14.87 seconds mean recovery time. The theoretical maximum of 110.27 seconds is reached when all services are contained in one group without colocation and monitored at an interval

time of 1 second. According to these values, the recovery time can be reduced by the factor 7.5.

$$RecoveryTime = 117.07 - 4.45 \times Timeout - 38.80 \times GroupingType + 36.45 \times ColocationType + ErrorTerm \quad (2)$$

V. Conclusion

In this paper we were interested in improvement of failover software which is used in cloud systems to satisfy "High Availability" requirements. We asked how we can make the Pacemaker software faster and described the "Recovery Time Test" as a procedure to find out what is causing slow recovery. We investigated how the Pacemaker configuration - the "Cluster Information Base" (CIB) - can influence Pacemaker recovery. We wanted to know 1) if it is possible to reduce the recovery time significantly by using a particular CIB configuration, 2) which CIB configurations are able to lead to decreased recovery time and 3) the factor by which we are able to reduce recovery time. In order to answer these questions, we applied the Recovery Time Test to the Pacemaker software in an OpenStack environment. We tested 24 CIB configurations by simulating outages and measuring the Pacemaker recovery time. We performed 30 test runs per configuration. We grouped all test runs under the same CIB configuration in samples and applied the Kruskal-Wallis H-Test to compare mean recovery times. We found out that there are significant differences between different samples. Therefore recovery time can be effectively reduced. We applied a Kruskal-Wallis MC-test in order to find configurations which lead to significantly lower recovery time compared to the others. The conclusion of these tests is that Pacemaker should manage OpenStack services as independent atomic resources rather than as one monolithic group. In contrast to that finding, colocation of OpenStack services on a single node has a decreasing effect on recovery time. We performed a (linear) regression using the CIB configuration parameters as coefficients and found a model for the recovery time. We used that model to evaluate recovery time reduction potential of CIB configurations and we saw that Pacemaker recovery time can be reduced by a factor of 7.5.

VI. Future work

These findings suggest to design cloud clusters that consist in ungrouped but strongly colocated services. The observation that services should be placed on one node but not managed as a large group seems to be paradox, but it becomes obvious when we consider that network communication is involved in the failover process. When it comes to a reboot procedure, the restart of a group of services is slower than the restart of individual services because the failover process is coordinated over the network which acts as a bottleneck. Recovery of colocated services is fast because it does not involve network communication. Future research should target analysis of dependencies between OpenStack services in terms of colocation and grouping. A well-colocated cloud architecture could be similar to multimaster replication for databases (e. g. the MySQL Galera cluster [32]). Such an architecture enables colocation of services with dependent data and could be a promising approach for the future.

References

[1] Tech Republic (2013). "What high availability for cloud services mean in the real world." http://www.techrepublic.com/blog/the-enterprise-cloud/what-high-availability-for-cloud-services-means-in-the-real-world/ Accessed on 2013/07/26.

[2] Marcus, E., Stern, H. (2003). *Blueprints for High Availability*. Indianapolis: Wiley Publishing.

[3] Perkov, L., Pavkovic, N., Petrovic, J. (2011). "High-Availability Using Open Source Software." *MIPRO 2011, Proceedings of the 34th International Convention*: 167-170.

[4] Haas, F. (2012). "Ahead of the pack: the pacemaker high-availability stack." *Linux Journal*, Vol. 2012 (Issue 216), 4.

[5] Loschwitz, M. (2011). "HA-Serie Teil 1: Grundlagen von Pacemaker und Co." *ADMIN*, Vol. 2011 (Issue 4).

[6] Youseff, L., Butrico, M., Da Silva, D. (2008). "Toward a unified ontology of cloud computing." *Grid Computing Environments Workshop*.

[7] Pepple, K. (2011). *Deploying OpenStack*. Sebastopol: O'Reilly Media.

[8] Baset, S. A., Tang, C., Tak, B. C., Wang, L. (2013). "Dissecting open source cloud evolution: An openstack case study." *USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, Vol. 13.

[9] Wen, X., Gu, G., Li, Q., Gao, Y., Zhang, X. (2012). "Comparison of open-source cloud management platforms: OpenStack and OpenNebula." *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*: 2457-2461. IEEE.

[10] Netflix (2010). "5 Lessons We've Learned Using AWS." http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html Accessed on 2014/04/01.

[11] Taylor, Z., Ranganathan, S. (2013). *Designing High Availability Systems: DFSS and Classical Reliability Techniques with Practical Real Life Examples*. Hoboken: Wiley-IEEE Press.

[12] Von Laszewski, G., Diaz, J., Wang, F., Fox, G. C. (2012). "Comparison of multiple cloud frameworks." *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*: 734-741. IEEE.

[13] Wu, Y., Huang, G. (2013). "Model-based high availability configuration framework for cloud." *Proceedings of the 2013 Middleware Doctoral Symposium*: 6. ACM.

[14] Berl, A., (et al.) (2010). "Energy-efficient cloud computing." *The Computer Journal*, Vol. 53 (Issue 7): 1045-1051.

[15] Clusterlabs (2013). "Pacemaker Architecture." http://clusterlabs.org/doc/en-US/Pacemaker/1.0/html/Pacemaker_Explained/s-intro-architecture.html Accessed on 2013/10/01.

[16] Hastexo (2012). "Highly Available Cloud: Pacemaker integration with OpenStack." http://www.hastexo.com/resources/presentations/highly-available-cloud-pacemaker-integration-openstack Accessed on 2013/09/26.

[17] OpenStack Foundation (2013). "The Pacemaker Cluster Stack." http://docs.openstack.org/high-availability-guide/content/ch-pacemaker.html Accessed on 2013/10/10.

[18] OpenStack Foundation (2013). "Setting up Corosync." http://docs.openstack.org/high-availability-guide/content/_setting_up_corosync.html Accessed on 2013/10/10.

[19] OpenStack Foundation (2013). "Starting Pacemaker." http://docs.openstack.org/high-availability-guide/content/_starting_pacemaker.html Accessed on 2013/10/10.

[20] Benz K., Bohnert T. M. (2013). "Dependability Modeling Framework: A test procedure for High Availability in Cloud Operating Systems." *Proceedings of the 1st International Workshop on Cloud Technologies and Energy Efficiency in Mobile Communication Networks (CLEEN)*.

[21] Netflix (2013). “Chaos Home.” https://github.com/Netflix/SimianArmy/wiki/Chaos-Home Accessed on 2013/06/01.

[22] Lilja, D. J. (2005). *Measuring computer performance: a practitioner's guide*. Cambridge: Cambridge University Press. 43-57.

[23] Tamai, T., Torimitsu, Y. (1992). “Software lifetime and its evolution process over generations.” *Proceedings of the IEEE Conference on Software Maintenance*: 63-69.

[24] Li, Z., Liang, M., O'Brien, L., Zhang, H. (2013). “The Cloud's Cloudy Moment: A Systematic Survey of Public Cloud Service Outage.” *arXiv preprint arXiv*:1312.6485.

[25] Chambers J. M., Freeny A. E. and Heiberger R. M. (1992). “Analysis of Variance: Designed Experiments.” *Statistical Models in S*, Chambers, J. M., Hastie T. J. (editors), Pacific Grove, CA: Wadsworth & Brooks/Cole.

[26] Marsaglia, G., Tsang W. W. and Wang J. (2003). “Evaluating Kolmogorov's distribution.” *Journal of Statistical Software*, Vol. 8 (Issue 18): 1-4.

[27] Levene, H. (1960). “Robust Tests for Equality of Variances.” *Contributions to Probability and Statistics*, Olkin, I. (editor), Palo Alto, CA: Stanford Univ. Press.

[28] Kruskal, W. H., Wallis, W. A. (1952). “Use of ranks in one-criterion variance analysis.” *Journal of the American Statistical Association*, Vol. 47 (Issue 260): 583-621.

[29] Chambers J. M. (1992). “Linear Models.” *Statistical Models in S*, Chambers, J. M., Hastie T. J. (editors), Pacific Grove, California: Wadsworth & Brooks/Cole.

[30] “1.3.3.24 Quantile-Quantile Plot.” www.itl.nist.gov/div898/handbook/eda/section3/qqplot.htm Accessed on 2014/11/12.

[31] Siegel S., Castellan N. J. (1988). *Non parametric statistics for the behavioural sciences.* New York: MacGraw Hill Int. 213-214.

[32] Codership (2013). “Galera Cluster for MySQL.” http://codership.com/content/using-galera-cluster Accessed on 10/10/2013.