Deriving Utility from a Self-Healing Benchmark Report

Simulator Architecture

Ritika Virmani, Rean Griffith, Gail Kaiser
{rv2171, rg2023, kaiser@cs}columbia.edu

Table of Contents

Problem Statement

Autonomic systems, specifically self-healing systems, currently lack an objective and relevant methodology for their evaluation. Due to their focus on problem detection, diagnosis and remediation any evaluation methodology should facilitate an objective evaluation and/or comparison of these activities. Measures of "raw" performance are easily quantified and hence facilitate measurement and comparison on the basis of numbers. However, classifying a system better at problem detection, diagnosis and remediation purely on the basis of performance measures is not useful. The proposed evaluation methodology devised will differ from traditional benchmarks, which are primarily concerned with measures of performance. In order to develop this methodology we rely on a set of experiments which will enable us to compare the self-healing capabilities of one system versus another. As currently we do not have available "real" self-healing systems, we will simulate the behavior of some target self-healing systems, system faults and the operational and repair activities of target systems. Further, we will use the results derived from the simulation experiments to answer questions relevant to the utility of a benchmark report

Statement of Work

Self-Healing systems are expected to reduce the cost and complexity of maintaining a system as the system itself takes care of any problems it encounters. They are expected to detect a system fault, perform a diagnosis and apply a remediation strategy. Work is going on to make self-healing activities greatly autonomic such that systems are able to "heal" themselves with minimal human intervention – and hence lower cost and time required for remediation. As we move towards making self healing systems better we need an effective benchmark to evaluate the efficacy of current self-healing systems. While the efficacy of vanilla systems (non-self healing) can be evaluated with easily quantifiable performance/throughput measures, self-healing systems are not as easy to evaluate and compare. A self-healing system is compared based on various factors such as the amount of time it takes to detect, diagnose and repair a fault, the cost and performance overhead , the accuracy of the repair strategy etc. These measurements are complex and a well-thought methodology needs to be applied in order to compare different self-healing systems based on these parameters. Hence, the need for a benchmark report to evaluate self-healing systems. Moreover, in order to derive utility from the benchmark report we need to evaluate which aspects of the benchmark report are useful.

This semester, I intend to design a simulator which will help us mimic the behavior of self-healing systems. The simulator will help carry out a step by step evaluation. For simplicity

we assume the following:

- Fault-independence i.e. no cascading faults
- Remediations are considered reactive
- Fault-diagnosis accuracy is 100%
- Fault-remediation accuracy is 100% with no negative side-effects.
- Remediations are single-step and fully automatic

Experiments

All experiments will be based on a time interval basis – defined as an "injection slot" [1] The simulator would incorporate the classic workload – disturbance – output model [2]. The workload model maps to a specific distribution defined for a system. For example consider a web application server dedicated to the stock market real time quotes. It would experience higher traffic during the mornings and evenings.

In the injection slot the system under test (SUT) first maintains a "steady state". A disturbance is then injected and the system performance drops to that defined by the "Disturbance" in the Response model. After a measured lag (the time required for the self-healing mechanism to begin remediation –system specific) the system output switches to the "repair" response.

The fault model and response model would define the context agnostic and context sensitive characteristics of faults and remediations as per our taxonomy:

Further the faults and remediation techniques are mapped defining the possible remediations for a fault. Similarly each remediation is mapped to all the faults it can fix.

Lastly, the simulator is concerned with measurements – both micro and macro. Micro measurements are centered on remediation, diagnosis etc. Macro measurements are concerned with the time and cost overhead and savings.

Each step is carried out with an objective as defined below. With each round, one or more of our assumptions cited above is relaxed.

Round #1

In round one a target system will be translated according to our methodology such that it facilitates our simulation experiments. We attempt to ascertain that the simulator works as desired. Also focus is laid on the interpretation of the resulting report. We refine out

measurement parameters if need be

Round #2

We evaluate systems on the basis of comparison in this round. First we compare a self-healing system to its plain-vanilla (non-self-healing) version. Next we take two versions of a self-healing system and compare them on the basis of their remediations for the same faults and the same work-load model (i.e in our definition – same context). Here, we also make our remediations more complex by assuming that they are not 100% accurate and single-step

Round #3

Focuses on more advanced simulations where we selectively relax more of our simplifying assumptions. Example, we consider cascading faults, fuzzy diagnosis and/or remediations. Further, we discuss approaches to evaluating self-healing systems that employ proactive and preventative remediations

The advantages of designing a simulator for the purpose of benchmarking self-healing systems are many. Exhaustive measurements can be made for several iterations of the experiments. The more exhaustive the measurements the more comparisons we can draw. Simulation will also help us model self-healing capabilities which have yet not been realized in real world systems. The top down approach we follow in the series of simulation experiments will help us derive utility from the benchmarking reports.

The potential benefits of deriving utility from benchmarking self-healing systems are manifold. The call for increased autonomic capabilities of systems calls for development of systems which can automatically repair faults manifesting in the system. Since any additional functionality invariably brings with it an overhead, the industry is concerned about the efficacy and efficiency of self-healing systems. A comprehensive benchmark report focused solely on this concern will spell out the potential advantages in quantifiable terms. It will serve to answer the questions about the feasibility and efficacy of any self-healing system.

What are we measuring?

Measure the performance of Self-Healing Systems in terms of:

- Availability : This is a measure of how many time units the system is available for (as per the SLA defined in the system model) work out of the entire simulation run

- Response Time : The time taken to process an incoming work request. We measure the minimum, maximum and average response times
- Throughput : The number of work requests processed per time unit during the entire run
- Repair Automation : A ratio of the number of faults repaired automatically by the system against the total number of faults injected in the system
- Work Breakdown : A breakup of the number of time units spent by the system in carrying out the following activities : processing work requests, detecting faults, diagnosing faults and repairing faults
- SLA adherence: Compliance to the SLA's defined for the system with respect to availability, response time and remediation time

Architecture

The simulator is implemented in Java. At the core of the simulator is the driver that behaves almost like a scheduler.

The simulator takes as input 4 models:

1. The Work Request Model.

The is modeled as an XML file as shown below: The work request XML defines the work requests that the system receives at each time unit during the simulation run.

```
<req>
<time>1</time>
<value>18</value>
</req>
```

Each work request node has a time and a value. The example above means that there were 18 work requests at time unit 1.

The work request model can be used to model various patterns of input. For example, we could test the system behavior against a step, ramp or impulse input

2. The Fault Model

The fault model is modeled as the XML file shown below. It defines the faults injected into the system during the simulation run.

```
<fault>
<id>MemoryLeak</id>
<severity>3</severity>
<time>9</time>
<time>79</time>
</fault>
```

Each fault node has an id which identifies the fault. It also has a severity on a scale of 1 to 4, 1 being the most severe. Then there are a number of child nodes with all the time instants at which that fault is injected

3. The Remediation Model

The remediation model is an XML file with nodes as shown below. Remediations are

context sensitive as defined in the paper. The context here is the system model and the fault. So a remediation such as restart carried out by system A for a fault f_i could be different for a fault f_j.

```
<remediation>
<id>MicroReboot</id>
<detectiontime>1</detectiontime>
<detectionimpact>1</detectionimpact>
<diagnosistime>1</diagnosistime>
<diagnosisimpact>1</diagnosisimpact>
<repairtime>1</repairtime>
<repairimpact>3</repairimpact>
<fault>MemoryLeak</fault>
</remediation>
```

So each remediation node has an id which identifies the remediation. It defines the time taken for detecting the fault and the overhead of the detection. Similarly it defines the time and overhead for the diagnosis and the repair. It then has a list of faults (fault ids from the fault XML) that this remediation is good for. A fault may have multiple remediations, in which case the simulator just picks the first one.

4. The System Model

The system model defines the target system itself. The following characteristics of the system are modeled:

1. Normal System Operation : Defines how many work requests the system processes per time unit under normal operation. For example we have taken it as 3 work requests per time unit for the sample runs that follow
2. SLAs
    1. Availability : Defines how many work requests the system should process per time unit to be considered available per the SLA. For example if the normal system operation is 3 WR / TU, the availability SLA has been taken as 2 WR/TU for the sample runs.
    2. Response Time : Defines the number of time units that every work request should be processed in.
    3. Remediation Time : Defines the number of time units that every remediation should be completed in.

Sample Results

Results from 2 sample runs of the simulator are given below.

- Baseline system with no self healing capabilities
- Self healing system.

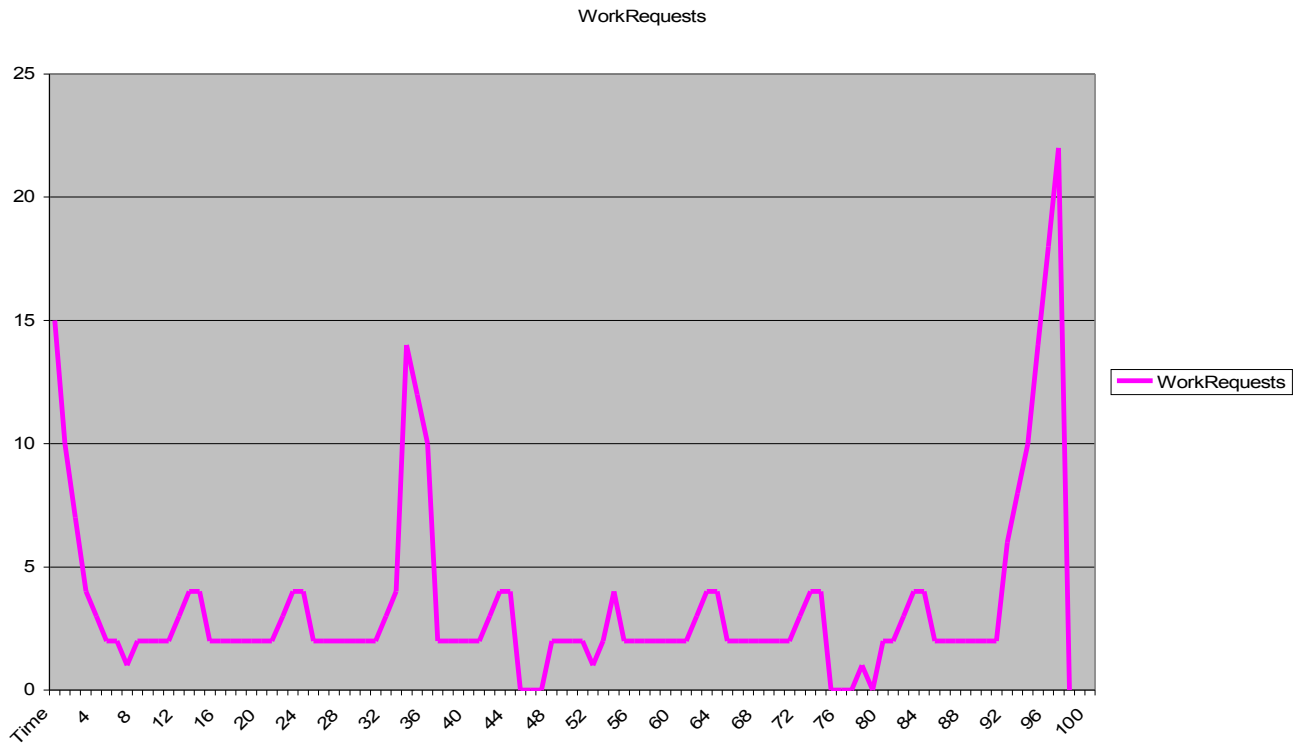
The **same** work request model, fault model and system model is used for both. So the exact same workload is applied to both the systems and the same faults are injected at the same time

We only use a **different** remediation model in each since in the baseline system the remediations are manual and in the self-healing system most of them are automatic.

Assumption for baseline system: While a personnel is busy repairing a fault that has occurred in the system, if another fault occurs, he only begins fixing it once the previous fault has been repaired

Input (Work Request)

The following graph shows the work request load applied as input. The input is analogous to a stock market with intense activity at the opening and close of the day. It also shows a spike during the middle of the day. Note that input is same for both the systems.



Results :

These graphs show the results play by play in the simulation run.

Legend:

- Blue color shows the work request processing.
- Purple color signifies detection
- Yellow color signifies diagnosis
- Cyan color signifies remediation

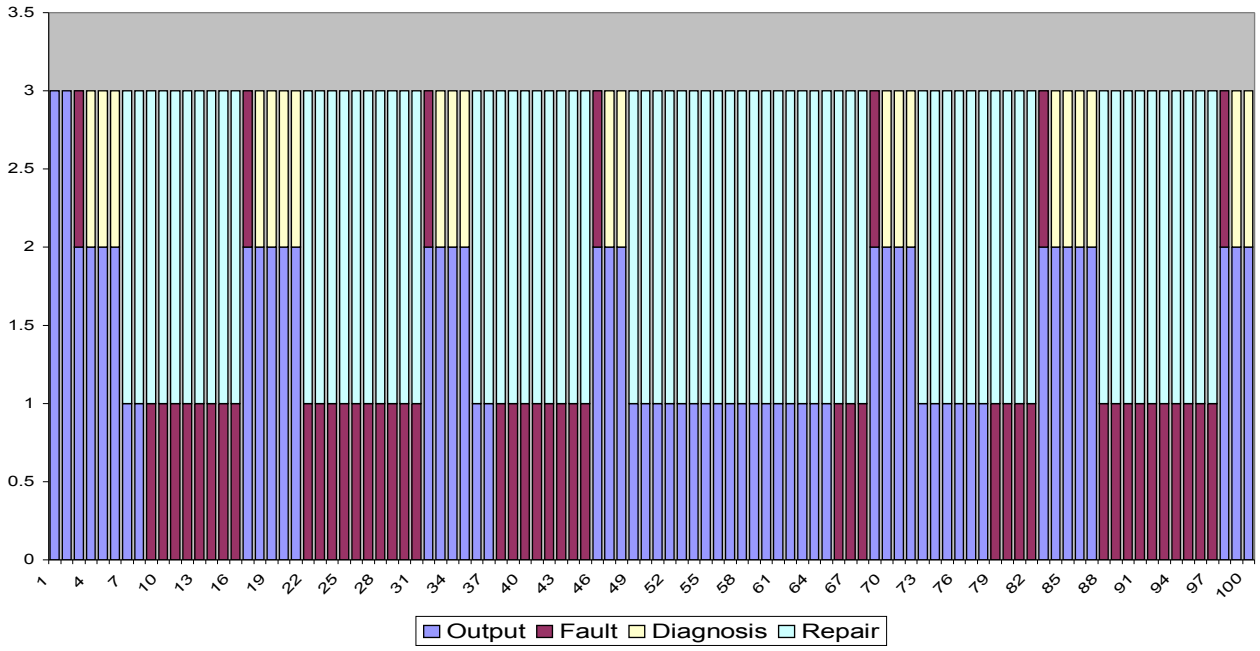
Points to Note:

- Note that the fault injection time in both cases is same, hence in the baseline system we see a lot more purple, since the system is personnel is still busy repairing the previous fault, the new fault just goes on impacting the system undetected.
- We see a lot of cyan in the first graph showing the amount of time the baseline system spends doing repair.
- We notice the cyan section in the middle of the graph for the self-healing system which is a manual repair called in due to a disk failure.

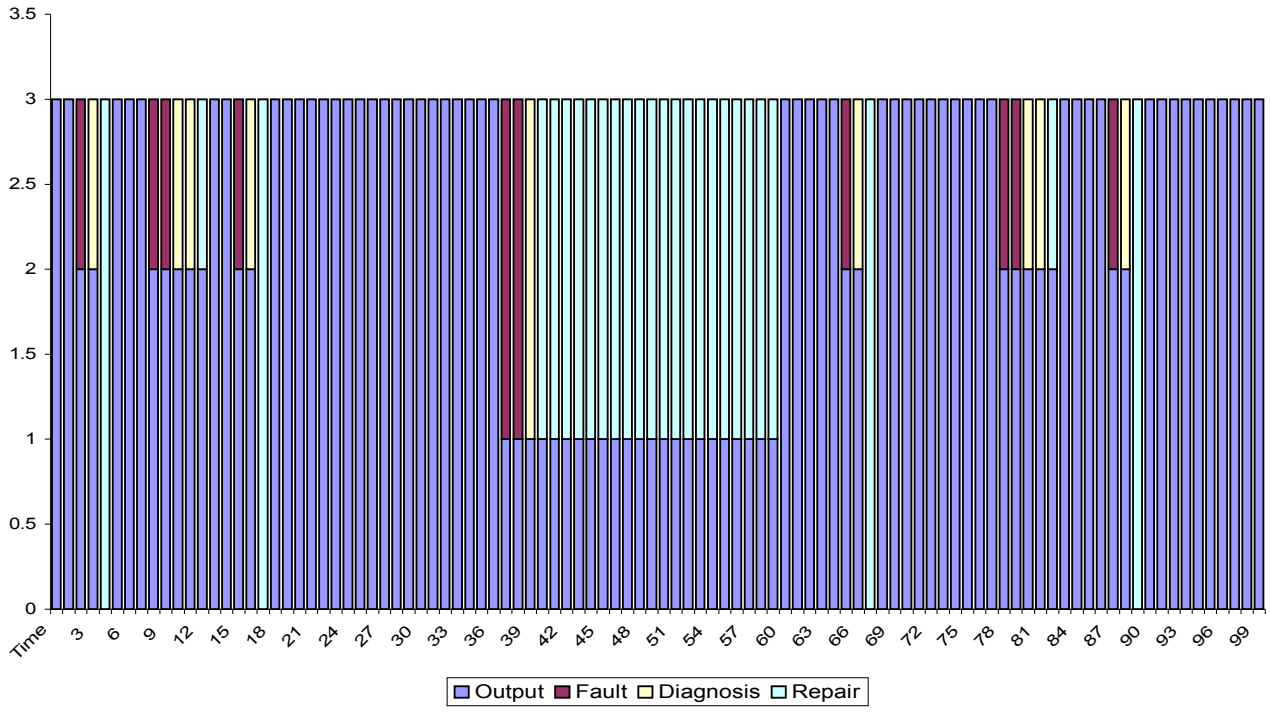
- The availability graph for the baseline system remains largely below the availability SLA line
- The availability graph for the self-healing system remains below the availability SLA line only when the system had to go in for a manual repair for a disk failure
- The availability for the baseline system is very poor as is evident from the large yellow section in the System Availability pie chart
- The availability of the self-healing is much better as seen in the System Availability pie chart
- The baseline system spends much of its time doing repair, shown by the large cyan section in the System Work Breakdown pie chart.
- In contrast the self-healing system spends much of its time in processing work requests shown by the large blue section in the System Work Breakdown pie chart.

Results Play by Play

Baseline System



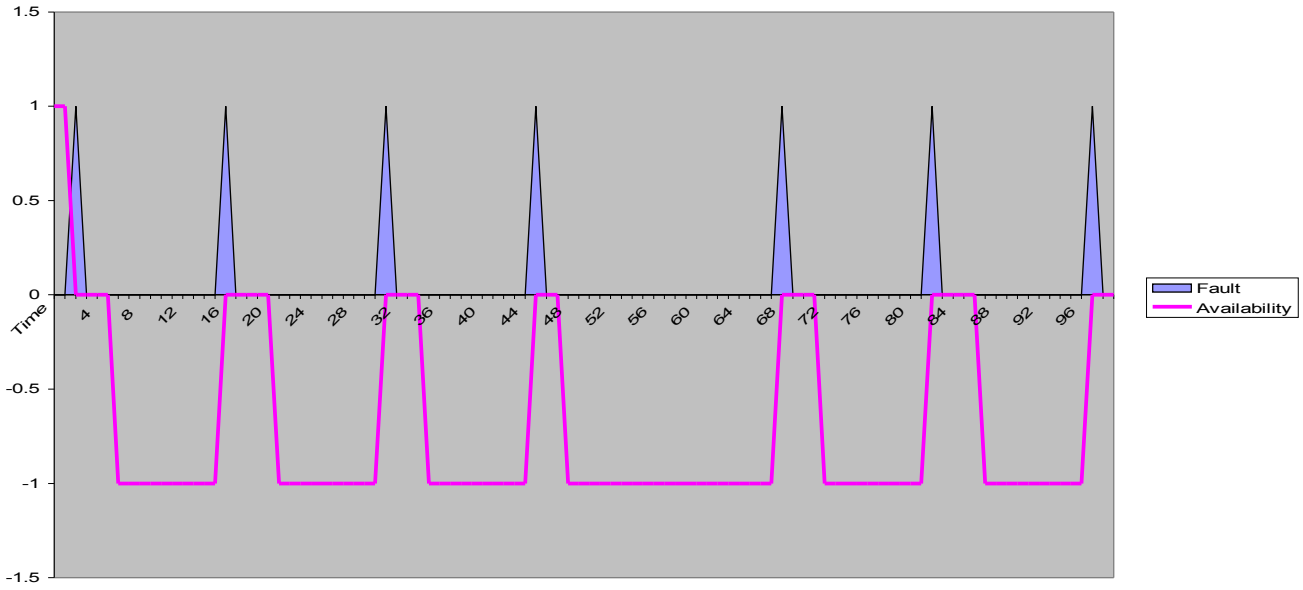
Self Healing System



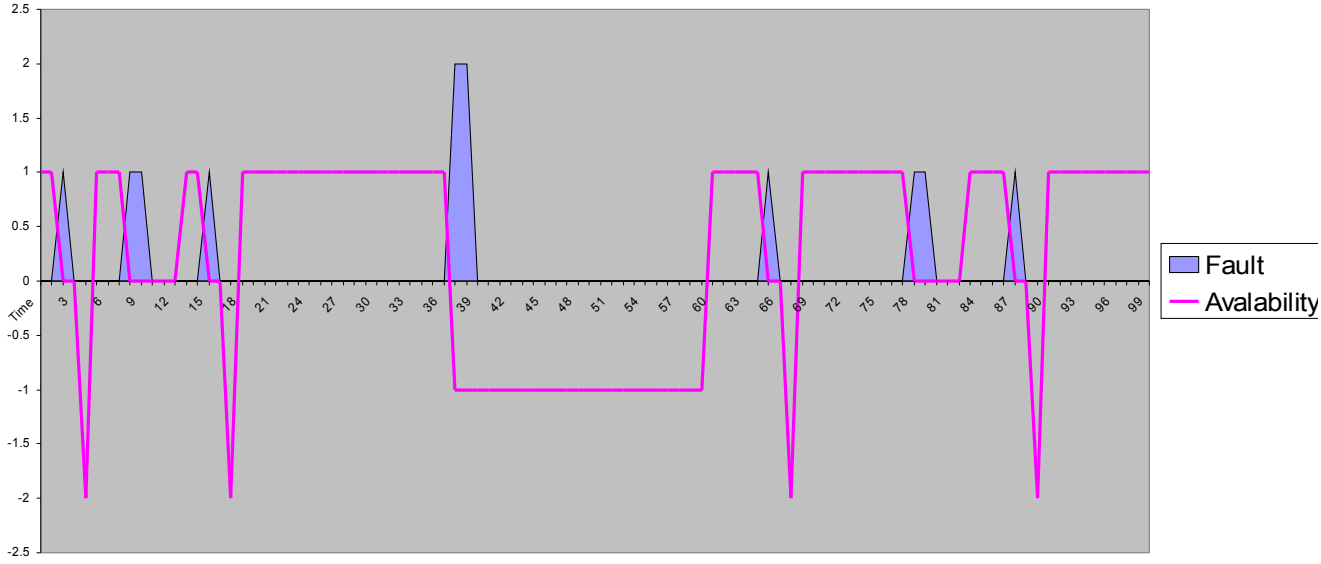
System Availability Graph with respect to the SLA

This graph has the SLA for availability as the baseline. The blue spikes show a fault injection. There is a stark contrast in the availability for both the systems

Baseline System



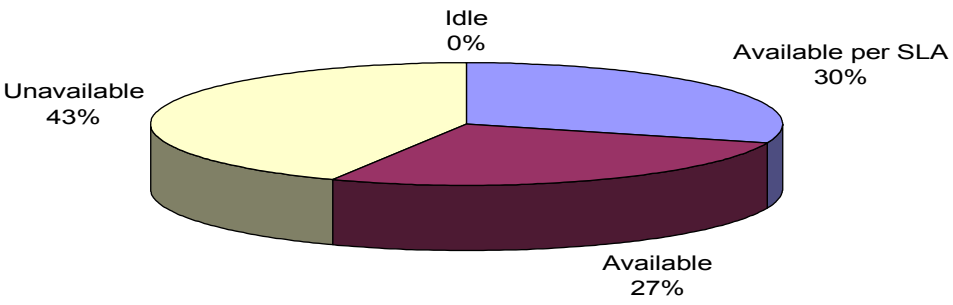
Self Healing System



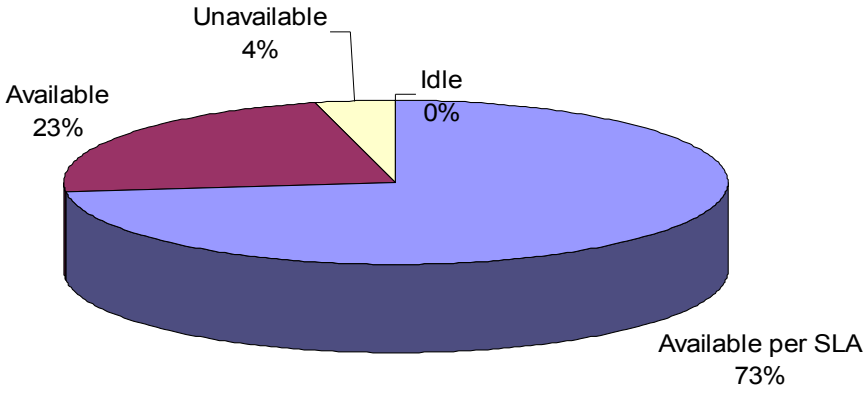
We notice the dip in availability in the center of the graph due to manual repair of the disk failure.

System Availability Pie Chart

Baseline System



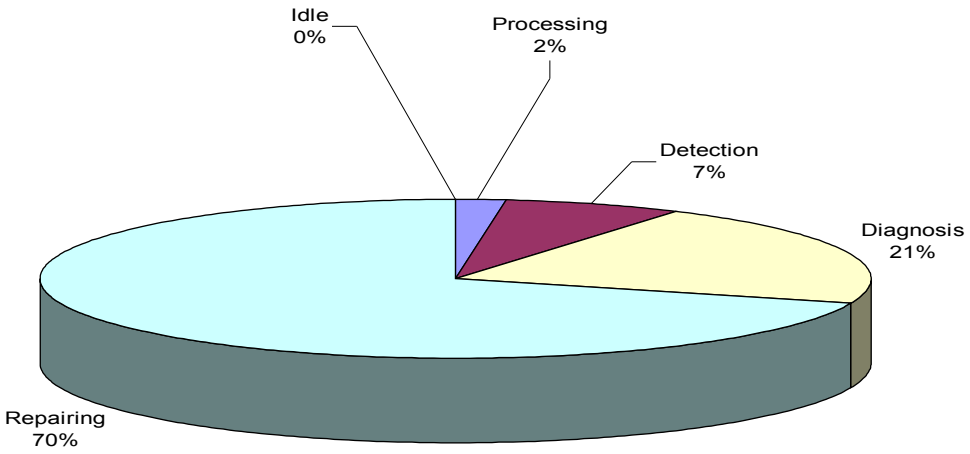
Self Healing System



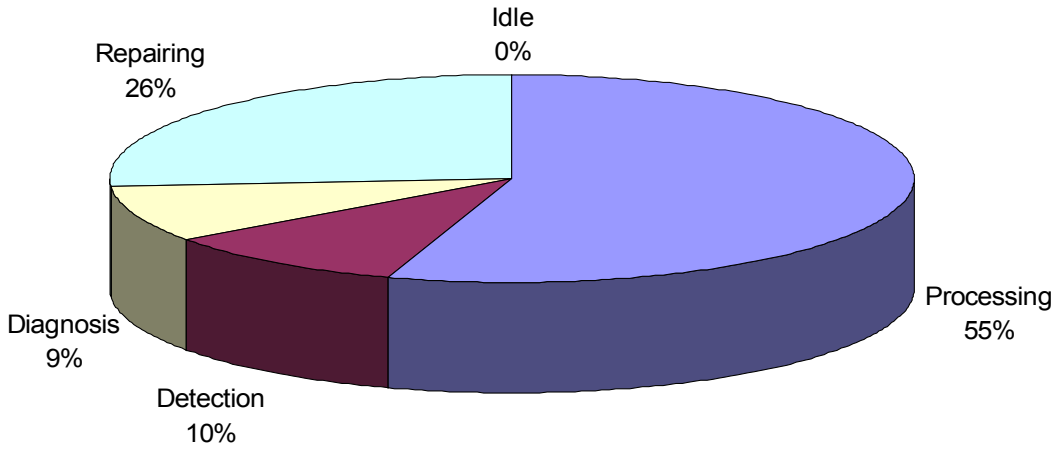
System Work Breakdown

This graph shows the amount of time the system spends in each of the following activities: work request processing, detection, diagnosis and repair. The baseline systems spends most of its time in repair

Baseline System



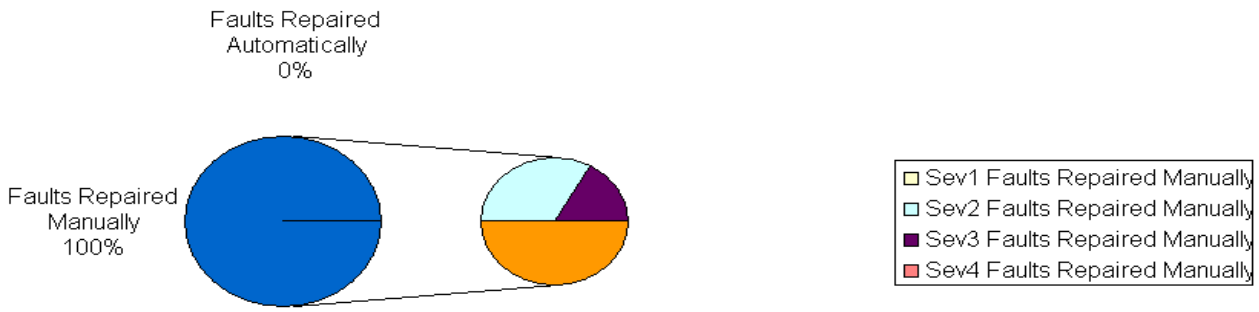
Self Healing System



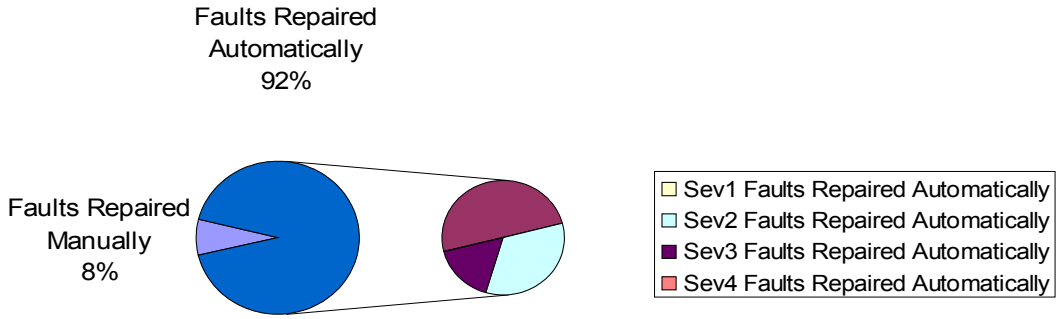
Repair Automation

This graph shows a comparison between the number of faults repaired automatically by the system against the ones that had to be repaired manually. For the faults that were repaired automatically, it shows the breakup of the repaired faults by severity. For the baseline system, all the faults are repaired manually.

Baseline System



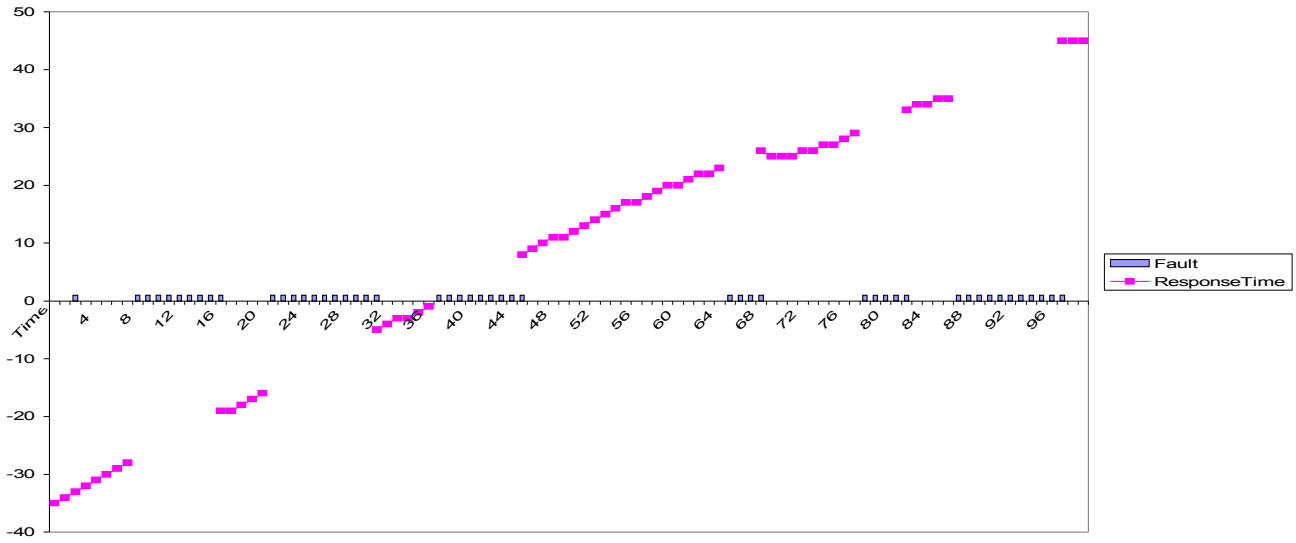
Self Healing System



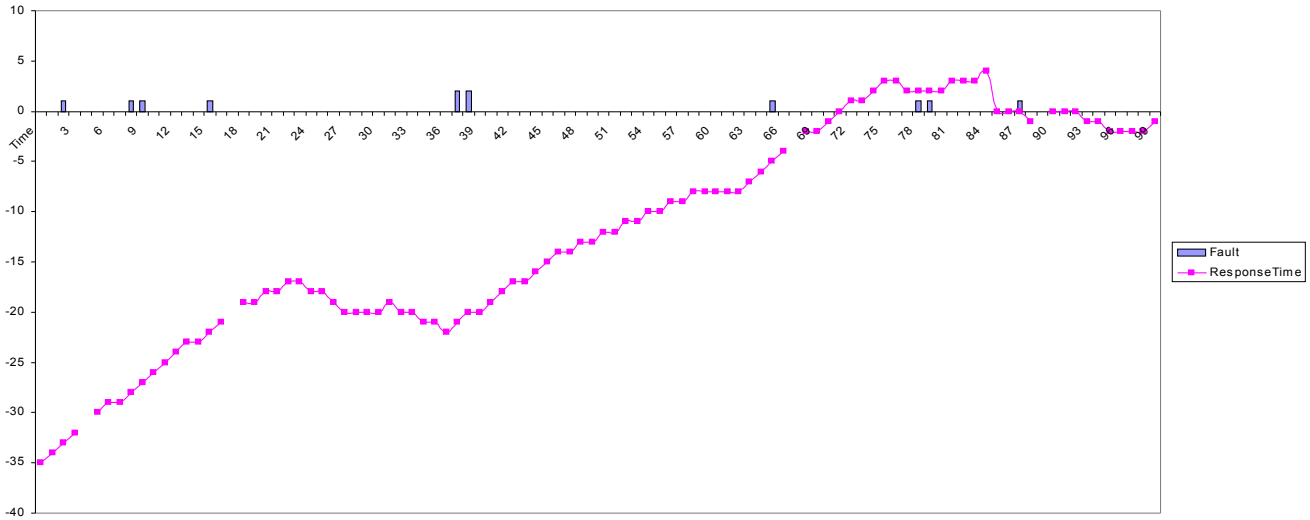
Average Response Time delta with respect to the SLA

The graph shows how the average response time varies during the course of the simulation with respect to the SLA. We notice that as we go through the simulation the response time rises and eventually goes over the SLA defined for it. The self-healing system regains stability with respect to response time, but for the baseline system, the response time goes on rising. Wherever the graph is broken, the system is unavailable

Baseline System



Self Healing System



More Results

More numbers to show the stark comparison between the two systems. There are clear contrast in every statistic

Baseline System

- **Availability=0.3**
- **Work Request Performance Measures**
  - Throughput=0.89 WR/TU
  - AvgResponseTime=41.73
  - MaxResponseTime=80
  - MinResponseTime=0
  - No of times ResponseTime SLA missed=53
  - % SLA met for Response Time=40.44%
- **Remediation Performance Measures**
  - Faults Repaired Automatically:0
  - Faults Repaired Manually:6
  - AvgRepairTime=22.33
  - MaxRepairTime=31
  - MinRepairTime=14
  - No of times Repair SLA missed=6
  - % SLA met for Repair Time=0.0%

Self Healing System

- **Availability=0.73**
- **Work Request Performance Measures**
  - Throughput=2.24 WR/TU
  - Avg Response Time=23.63
  - Max Response Time=39
  - Min Response Time=0
  - No of times ResponseTime SLA missed=41
  - % SLA met for Response Time=81.69%
- **Remediation Performance Measures**
  - Faults Repaired Automatically:6
  - Faults Repaired Manually:1
  - AvgRepairTime=6.42
  - MaxRepairTime=23
  - MinRepairTime=3
  - No of times Repair SLA missed=3
  - % SLA met for Repair Time=57.14%

Conclusion

From the results we clearly see the difference between the baseline system and the self-healing system. The statistics which bring out the difference help us determine which measurements help us in benchmarking self-healing systems.

We see here that the availability increases greatly for a self-healing system. The average response time becomes more stable. The percentage response time SLA almost doubles.

If manual repairs were attached a cost metric, we would see significant cost savings. Since 92% faults are repaired automatically by the self-healing system, our cost savings would be 92%.

Of course, we meet the repair SLA 92% of the time in the self-healing system.

Limitations

- Remediation Accuracy 100%
- No Cascading Faults
- One fault at a time
- Remediations are reactive

Acknowledgements

The Programming Systems Laboratory is funded in part by NSF grants CNS-0627473, CNS-0426623 and EIA-0202063, NIH grant 1U54CA121852-01A1, NYSTAR, Financial Services Technology Consortium, and Consolidated Edison Company of New York.

References

[1] A. Brown and P. Shum. Measuring resiliency of it systems. http://www.laas.fr/kanoun/WsSIGDeB/5-IBM.pdf.

[2] A. Brown and C. Redlin. Measuring the Effectiveness of Self-Healing Autonomic Systems. In 2nd International Conference on Autonomic Computing, 2005