# Recovery Oriented Computing in Distributed Systems

Vincent Borchardt

Department of Computer Science
University of Minnesota, Morris

November 30, 2012

# Traditional System Design

System designs are often built on unrealistic assumptions:

- Systems do not fail.
- When failures occur, they occur in isolation.
- Failures only occur when components fail.

Emphasis is placed on preventing failures rather than repairing failures.

# Realities of Systems

Realities of systems:

- Systems do fail in many ways.
- Failures occur in many combinations of components.
- Failures occur primarily when operators make errors.

More failures occur than expected.

Failures are difficult to repair.

# Costs of Failures

Well-managed systems generally are working 99% to 99.9% of the time–systems are down 8 to 80 hours a year.

Each hour a system is down costs between $200,000 (a commerce website) and $6,000,000 (a stock-trading system) depending on how critical the system is [3].

Decreasing the time the system is down can save millions of dollars a year.

# Overview

1. Introduction
   - Traditional System Designs
2. Fault Tolerance and Recovery Oriented Computing
   - Fault Tolerance Measurements
   - Recovery Oriented Computing
3. Distributed Systems
   - Grid Computing
   - Stream Computing
4. Conclusion
   - Comparing the Systems

1 Introduction
- Traditional System Designs

2 Fault Tolerance and Recovery Oriented Computing
- Fault Tolerance Measurements
- Recovery Oriented Computing

3 Distributed Systems
- Grid Computing
- Stream Computing

4 Conclusion
- Comparing the Systems

# Mean Time to Failure and Mean Time to Recovery

Mean Time to Failure (MTTF): Average time it takes for the given component/system to fail

- Example: Average time until a hard drive fails

Mean Time to Repair (MTTR): Average time it takes for the given component/system to be repaired/replaced

- Example: Average time to replace a hard drive

# Large-Scale MTTF and MTTR

Network File System (NFS) Server: Many hard drives linked together, then partitioned to provide consistent space for a user on multiple computers in a network.

Scenario: A single hard drive dies, which causes the system to fail:

- The MTTF of the hard drive is the average time until a hard drive fails, but the MTTF of the system is the average time until any failure occurs that causes the system to fail (any of the components fail, the software fails, a user makes an error).

# Large-Scale MTTF and MTTR

Network File System (NFS) Server: Many hard drives linked together, then partitioned to provide consistent space for a user on multiple computers in a network.

Scenario: A single hard drive dies, which causes the system to fail:

- The MTTR of the hard drive is the time it takes to replace the hard drive (buying a new one, installing it in the system), but the MTTR of the system includes all the steps until the system is running normally again for an average failure (rebooting the system, restoring from backups, confirming everything is working correctly).

# Large-Scale MTTF and MTTR

Network File System (NFS) Server: Many hard drives linked together, then partitioned to provide consistent space for a user on multiple computers in a network.

Scenario: A single hard drive dies, which causes the system to fail:

- The MTTF and MTTR for systems is much more complicated than that for individual components.

# Problems with a Prevention-Focused Approach

Estimation of MTTF of components is flawed:

- Based on a large group of components over a short time, but the failure rate is not independent of time

Example of MTTF estimation from [4]:

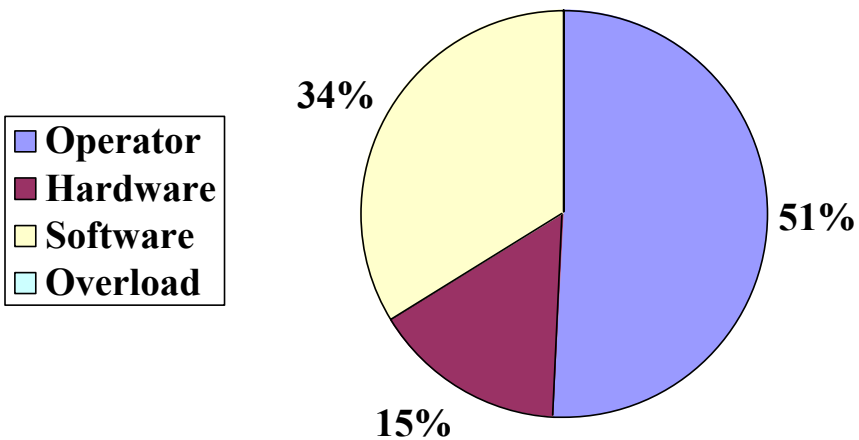
- 1,000 hard drives run for 3,000 hours: 3,000,000 operation hours total
- 10 drives fail–1 failure per 300,000 hours of operation
- MTTF = 300,000 hours $\approx$ 34 years

# Problems with a Prevention-Focused Approach

Systems already have a large MTTF.

Many failures are based on operator error, rather than hardware or software errors.

This means that having the individual components' MTTF be large is not as important for preventing failures.




Causes of website failures from [3]

# Uptime and Downtime

Uptime: % of the time the system is running (also called Availability)

$$\text{Uptime} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})}$$

Downtime: % of the time the system is not running

$$\text{Downtime} = 1 - \text{Uptime} = \frac{\text{MTTR}}{(\text{MTTF} + \text{MTTR})}$$

# Recovery Oriented Computing

Errors are "facts to be coped with, not problems to be solved." [3]

Availability is most important, not just MTTF.

MTTR is generally focused on to reduce downtime:

- If MTTR is small compared to MTTF, halving MTTR has the same impact as doubling MTTF:

$$\text{Downtime} = \frac{\text{MTTR}}{(\text{MTTF} + \text{MTTR})} \approx \frac{\text{MTTR}}{\text{MTTF}}$$

1 Introduction
- Traditional System Designs

2 Fault Tolerance and Recovery Oriented Computing
- Fault Tolerance Measurements
- Recovery Oriented Computing

3 Distributed Systems
- Grid Computing
- Stream Computing

4 Conclusion
- Comparing the Systems

# Distributed Systems in General

Multiple computers working together to accomplish a single goal.

These large systems have many components, which implies a low MTTF and high MTTR for the overall system.

Systems are classified by the types of problems they solve.

Two types covered:

- Grid computing
- Stream computing

# Introduction to Grid Computing

Components (computers) are tightly coupled with each other

Problems are split up among the components

Time-critical actions:

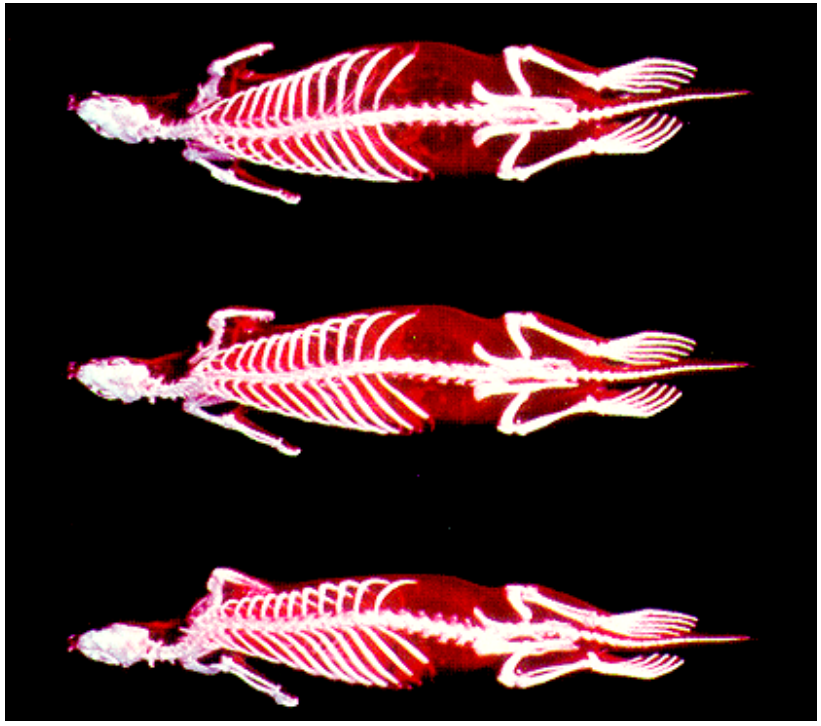
- Strict time constraints
- Focus on maximizing benefit (useful work done)
- Minimum amount of benefit required to make the task worth doing

# Tissue Volume Rendering

Real-time rendering and display of 2-D images from a 3-D data set

Goal is to produce a large, accurate image

Useful for doctors during surgery



A rendering of sea otters from [1]

# Benefit

Each action taken by the system has a benefit function $B$ that determines how well the action is doing its job.

- Volume Rendering: How large/accurate the image is

Each action also has a baseline benefit $B_0$ that represents the minimum level of acceptable benefit–if $B < B_0$, the action is not worth doing.

$$\text{Benefit Percentage} = \frac{B}{B_0} * 100\%$$

Each of these parameters are based on a time constraint $T_c$–benefit obtained outside of $T_c$ has little to no usefulness.

# Reliability

Each component of the system $N^i$ has a reliability value $R_N^i$, which says how likely the component is to fail.

- $R_N^i \in [0, 1]$, where 0 means the system always fails and 1 means the system never fails.

The possibility of failure goes up over time (which correlates to the MTTF of the system), and failures do not occur in isolation.

# Reliability vs Efficiency

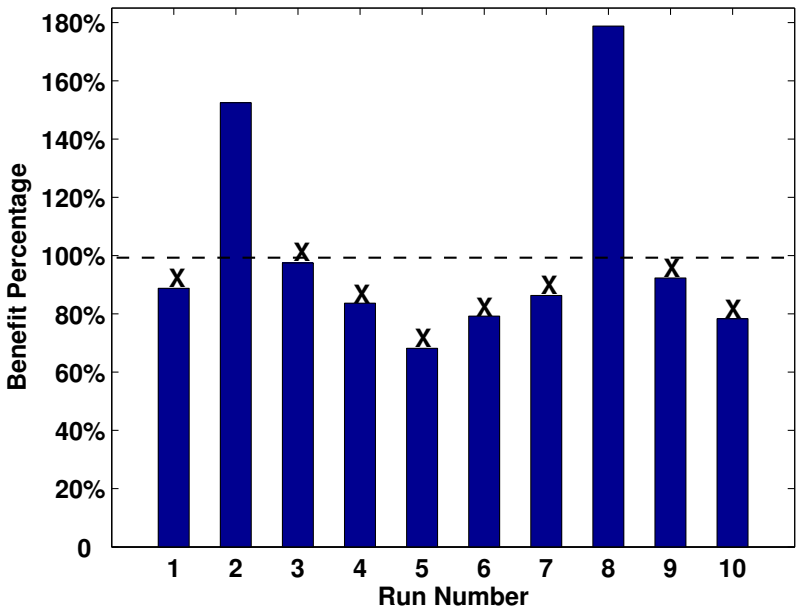
Available resources (computers) have varying amounts of reliability and efficiency.

In general, resources only excel in either reliability or efficiency.

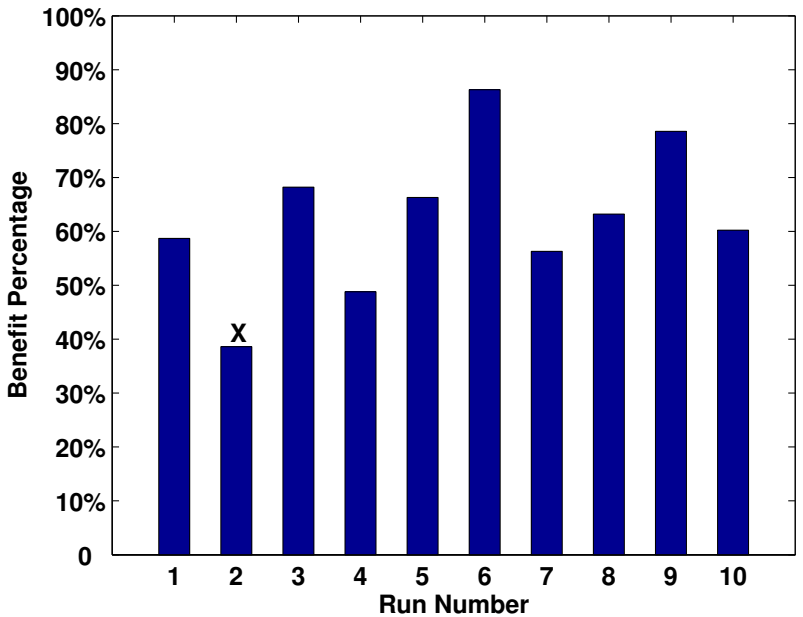
Choosing only reliable resources means the system can't meet the baseline efficiency.

Choosing only efficient resources means the system fails often enough so it can't meet the baseline efficiency.

# Reliability vs Efficiency




Test results of the Volume Rendering system from [5] using only efficient resources.




Test results of the Volume Rendering system from [5] using only reliable resources.

# Maximizing Benefit and Reliability

To maximize the performance and reliability of the system, the resources chosen must:

- Maximize benefit $B$
- Maximize reliability $R_N^i$
- Meet the baseline benefit $B_0$
- Stay within the time constraint $T_c$

Maximizing two variables ($B$ and $R_N^i$) is a difficult problem–an evolutionary algorithm is used to determine a good resource set.

# Failure Recovery: Replication

Even with reliable resources, failures still happen.

The simplest way to guard against failures is to replicate the entire system, but that has problems:

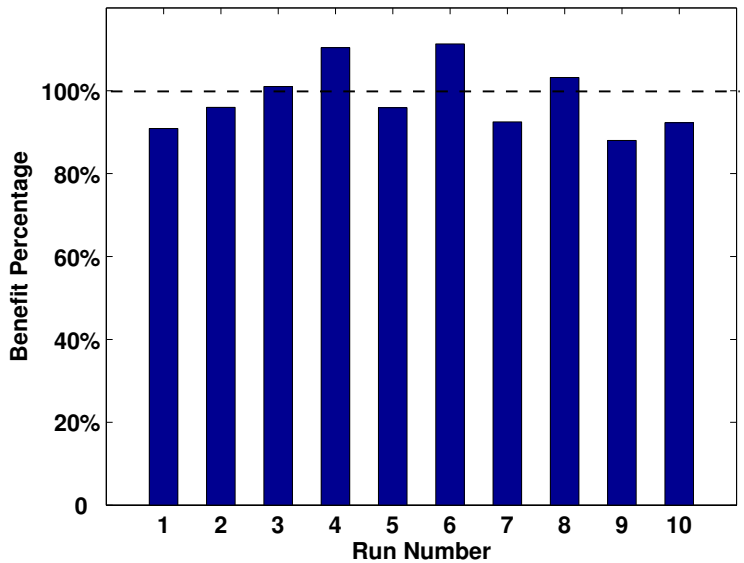
- Incurs significant overhead–very bad for time-critical operations
- Each duplicate needs to do the same thing–doesn't work in systems with randomness

# Replication Results

The graph shows the Volume Rendering system run with good resources and fault tolerance by replication:

- No failures
- Average benefit = 96%

The system still isn't meeting the baseline benefit on average.




From [5]

# Failure Recovery: Checkpointing

The duplicate systems exist, but aren't running concurrently with the main system.

Progress in the main system is periodically transferred to the backup systems.

The main system sends a heartbeat message periodically to all the backup units.

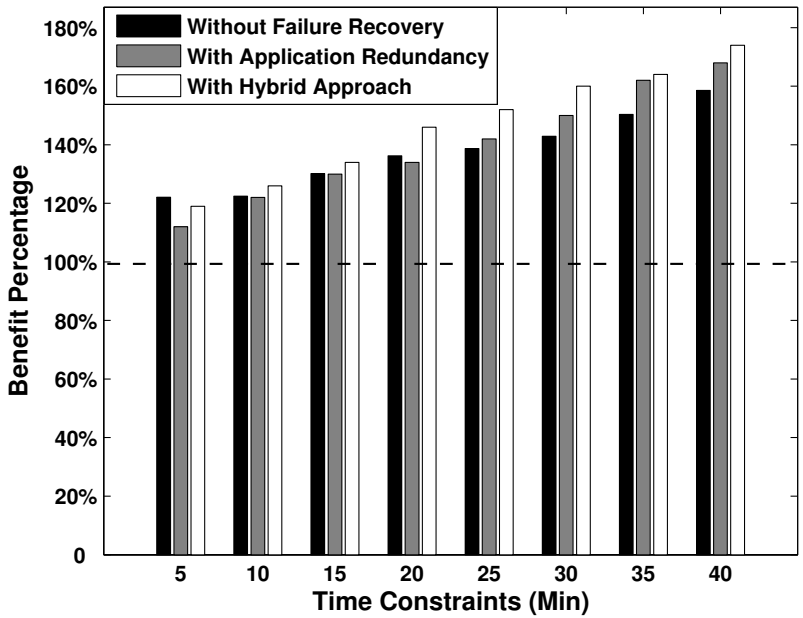
If a backup unit doesn't hear from the main system in a certain period of time, it assumes the main system has failed and takes over the action.

# Overall Results of Fault Tolerance

The graph shows benefit percentage for different time constraints using three variations of the Volume Rendering system:

- No failure recovery
- Replication only
- Replication and checkpointing

The system with checkpointing obtained the most benefit in the majority of cases, exceeded the replication only strategy in all cases, and exceeded the baseline benefit in all cases.




From [5]

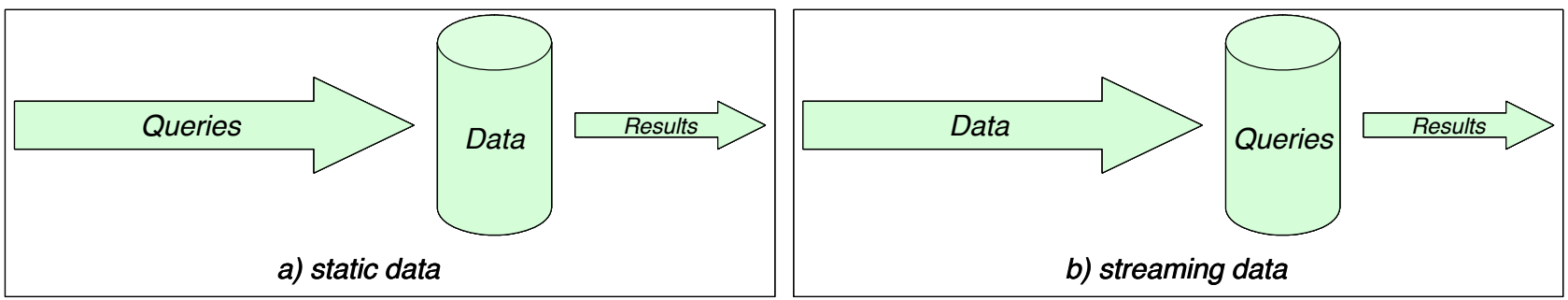
# Introduction to Stream Computing

Uses streaming data:

- Information is obtained continuously, without having to ask for that information multiple times

Example: GPS

- Static data: You ask the GPS for your location when you need that information.
- Streaming data: When your location changes, the GPS gives you your new location.



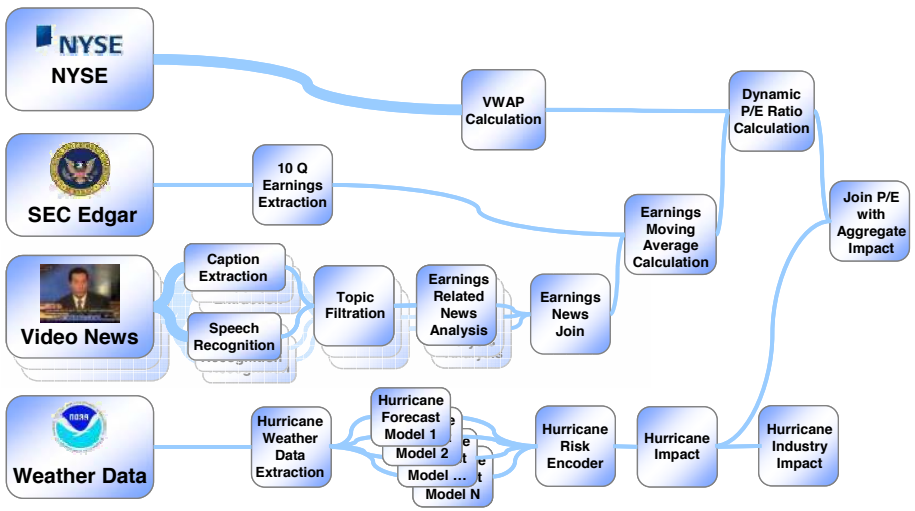

Visual representation of static versus streaming data from [2]

# System S

Gets data from many different sources

Example: Predicting a company's stock price:

- Many sources of information:
  - Past performance of the stock
  - Opinions of the news on the company
  - Impact of the weather forecast on the company
- Prediction changes as all of these sources change




Data streams used by a stock trading stream computing system from [2]

# Processing Streams in System S

Streams are processed by smaller parts of the system, then that information is processed by the larger system.

Scheduling the processing of a group of streams is known as submitting a job.

Submitting a job involves checking if the job can be executed, letting the larger system know that the job is being executed, then processing the data.

# Fault Tolerance in System S

Two major steps to submitting a job:

- Checking if the user has permission to submit a job
- Registering the job in the larger system and processing the data

A job may only be registered in the larger system once.

What happens if a job submission is interrupted because a failure occurred in the system?

- The smaller system does not know if the job has been registered in the larger system.
- The process cannot be restarted in a reasonable way.

# Fault Tolerance in System S

Two major steps to submitting a job:

- Checking if the user has permission to submit a job
- Registering the job in the larger system and processing the data

A job may only be registered in the larger system once.

Implementing fault tolerance for submitting a job:

- The two parts of submitting a job are split into two distinct actions: preparation and registration.
- After the permission is checked in the first action, an ID is assigned to the job.
- That ID will be used in the second action to actually register the job (like reserving a place in line).

# Fault Tolerance in System S

Two major steps to submitting a job:

- Checking if the user has permission to submit a job
- Registering the job in the larger system and processing the data

A job may only be registered in the larger system once.

What happens if this new process is interrupted by a failure?

- The first action doesn't change the state of the larger system, so it can be restarted at will.
- If the second action is interrupted, the job still has an ID, and that ID can be used to resume the process.

This isolates failures, ensures the system is always in a consistent state, and greatly reduces the time needed to recover from a failure.

# Summary

Recovery Oriented Computing techniques are implemented in different ways for different systems:

- The time-critical Volume Rendering grid system chose the right combination of efficient and reliable resources and reinforced that strategy with a combination of checkpointing and replication.
- System S used stream computing and was separated into small components so parts of the system could be restarted quickly.

# Comparison of the Systems

The difference in fault tolerance in the two types of systems comes from the way they solve their problems:

- The grid computing systems are tightly coupled and only communicate with the larger system, so the fault tolerant communication mechanisms between the smaller parts of the system are not worth the extra overhead.
- The available resources for a stream computing system are constantly changing, so it isn't appropriate for the stream computing system to spend time looking for good resources.

This shows that ROC techniques can be different for many different systems.

# Questions?


R. A. Drebin, L. Carpenter, and P. Hanrahan.
Volume rendering.
In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '88, pages 65–74, New York, NY, USA, 1988. ACM.


IBM.
System S - Stream Computing at IBM Research.
http://download.boulder.ibm.com/ibmdl/pub/software/data/sw-library/ii/whitepaper/SystemS_2008-1001.pdf, 2008.


D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tezzlaff, J. Traupman, and N. Treuhaft.
Recovery oriented computing (ROC): Motivation, definition, techniques, and case studies.
Technical report, U.C. Berkley, 2002.
http://roc.cs.berkeley.edu/papers/ROC_TR02-1175.pdf.


J. H. Saltzer and M. F. Kaashoek.
*Principles of Computer System Design Part 2*.
MIT Open Courseware.
Chapter 8, `http://goo.gl/hhppt`.


Q. Zhu and G. Agrawal.
Supporting fault-tolerance for time-critical events in distributed environments.
In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 32:1–32:12, New York, NY, USA, 2009. ACM.