RemusDB

Transparent High Availability for Database Systems

Presented by: Marcel Gietzmann-Sanders, Qianchao Nie

Outline

- Introduction and Motivation
- Normal High Availability Architectures
- Remus Overview
- ASC – Asynchronous Checkpoint Compression
- RT – Disk Read Tracking
- CP – Commit Protection
- Experimental Evaluation
- Conclusion

Introduction and Motivation

What is High Availability?

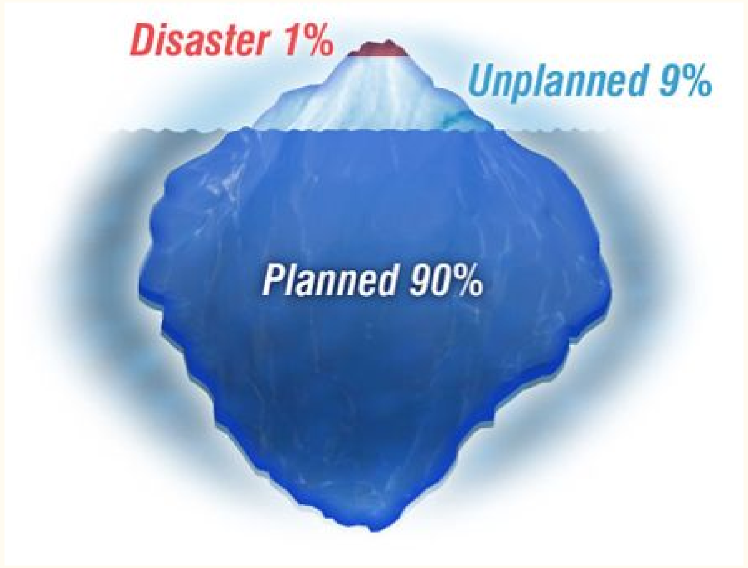
Computing environments configured to provide nearly full-time availability are known as high availability systems.

Introduction and Motivation

Why do we need to provide system and database High Availability?

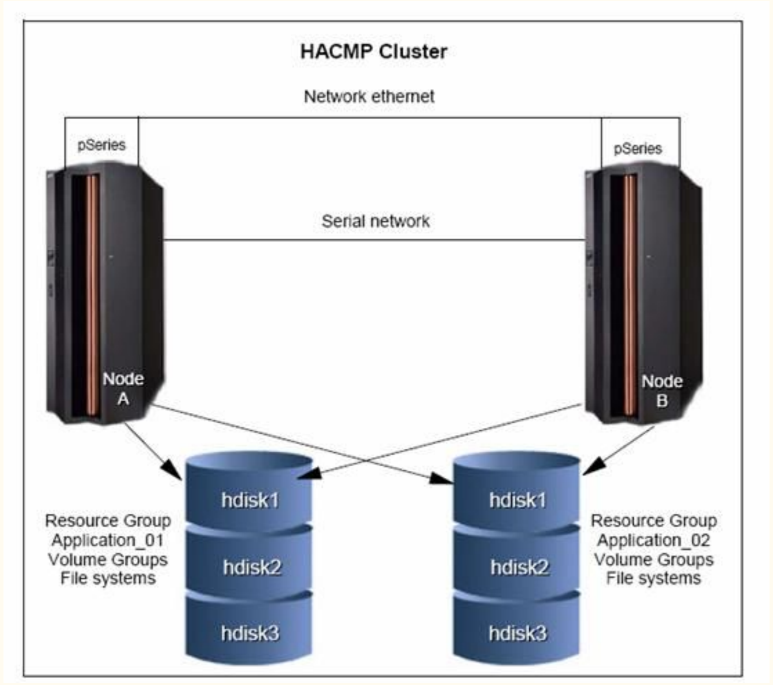
Planned Downtime
－Database backup/upgrade/patching
－Operating system upgrade/patching
－Hardware and Network maintenance

Unplanned Downtime
－Human Errors
  Accidentally drops a table
  Accidentally delete a data file or drop a tablespace
－Disasters
  Server crash, malfunction of hardware
  War, terrorism, Earthquake
  No power for a long period



Normal High Availability Architectures

OS High Availability


HACMP Cluster
Network ethernet
pSeries
pSeries
Serial network
Node A
Node B
hdisk1
hdisk2
hdisk3
hdisk1
hdisk2
hdisk3
Resource Group Application_01 Volume Groups File systems
Resource Group Application_02 Volume Groups File systems

Theory

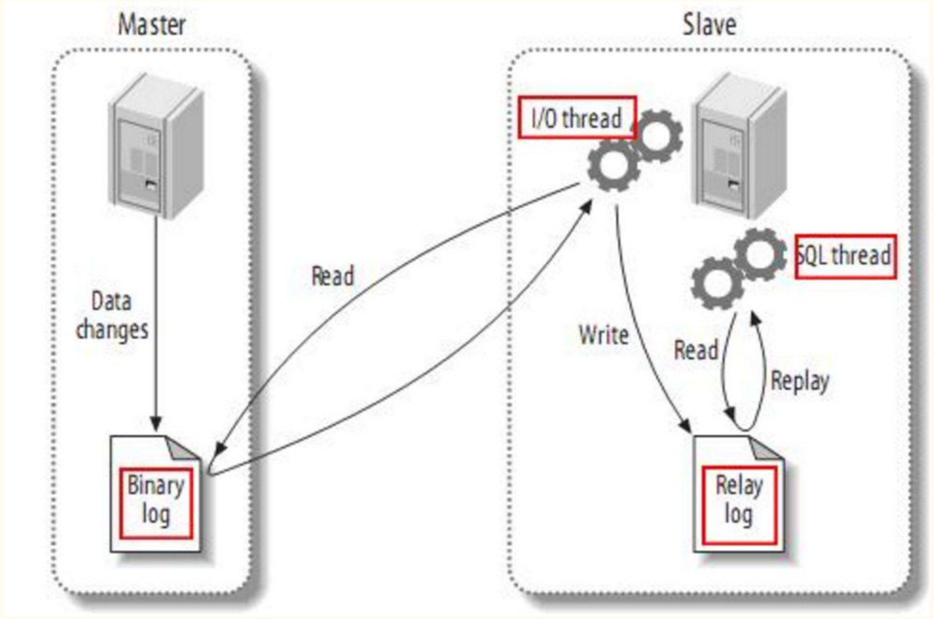
- Primary and standby server share same resource group.(Disk, IP address, File system...)
- During normal time, Primary hold the resource group and provide service to clients
- If there is a failover, standby server will take over the resource group and continue provide service to clients, consistency can be maintained.

Disadvantages:

- Not a warm-up backup, failover is not transparent to clients, and a relatively long time to recover

Normal High Availability Architectures

Database High Availability



Theory

- Backup database redo the transaction log sent by the primary database
- Because backup database is warm up, so the takeover time is short

Disadvantages:

- On failover, transaction log may lose
- Lose all existing client sessions at failure
- Complex to implement for DBMS and a lot of bugs have been reported

Normal High Availability Architectures

Some other advanced HA solutions:

- Oracle RAC
- DB2 PureScale
- ….

Disadvantages:

- Advanced commercial features are too expensive and require specific hardware

- Too complex to configure and maintain, different feature to different database package

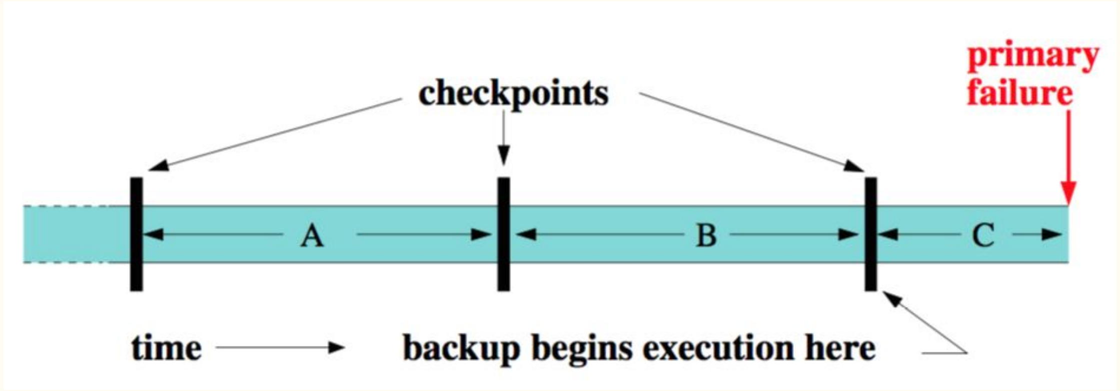
The Idea Behind Remus

Many DBMS applications are already run in VMs thanks to things like

- Live migration (transferring a virtual machine (VM) between physical computers without losing client connections or applications)
- Elastic Scaleout
- More flexible use of resources

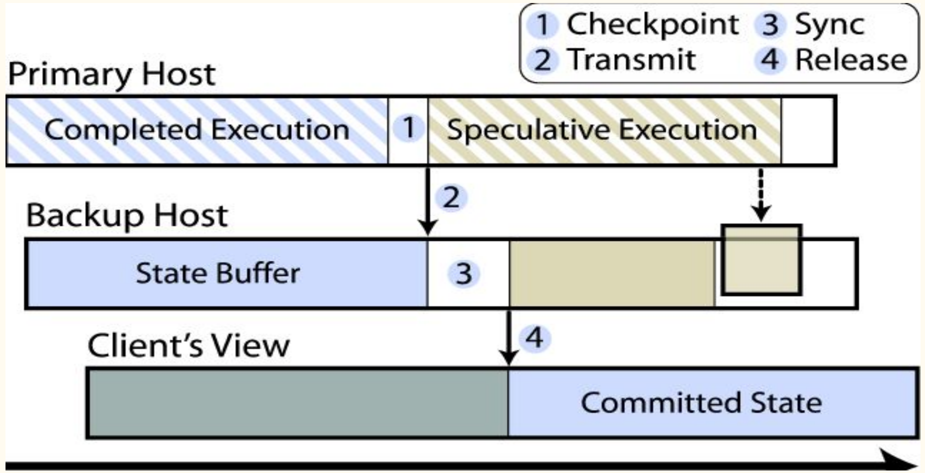
But because **VM's give us the power to copy any part of the VM's state**, running a DBMS in a VM allows us to **push the creation and maintenance of the standby out of the application layer and into the virtualization layer.** This is what **Remus** does for general applications (not just DBMS).

Remus Overview: Capturing State



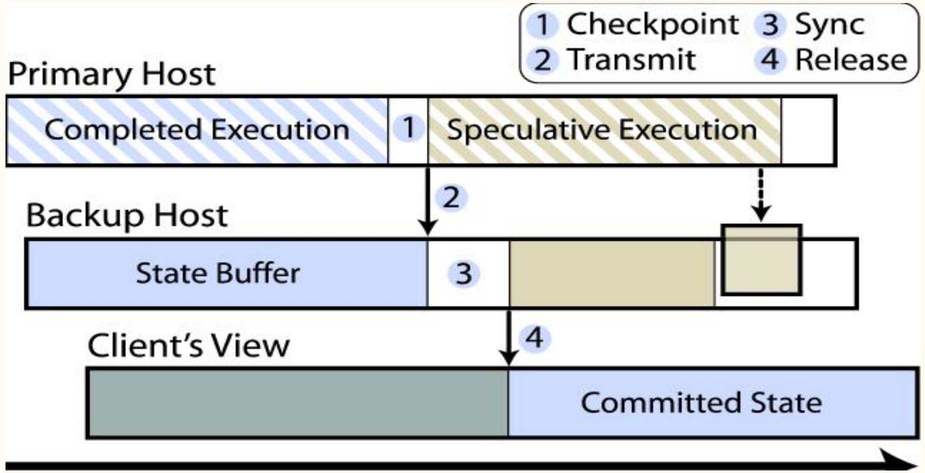
- Remus takes frequent **snapshots** of the **entire** active server's **VM state**. These snapshots are called checkpoints(as often as every 25ms).
- Each of the **snapshots get installed on the standby**.

Remus Overview: Handling of the Network



- **Between checkpoints** all outgoing **network packets are held back** by Remus until the checkpoint is complete.
- **If the active fails, Remus transparently transfers client connections to the standby** and simply continues using the state the standby currently has.

Remus Overview: Handling Disk



- **All writes** to the active's disk **are sent to a buffered pool on the standby**.
- This **pool is flushed to disk once per checkpoint** after all the state has been transferred from the active and the standby starts to sync.
- Remus **marks the standby's disk** to keep track of the **last consistent state** so that **in the case of failure there is a crash consistent disk image.**

Properties of Failover Using Remus

- **Because the entire state is copied** to the standby we have a warmed-up replica of the active as soon as the active fails. This means **the handover period is extremely small.**
- **Because Remus transparently transfers connections to the standby on failover, besides a small lag** during the handover itself, **the clients notice nothing**.
- **Because any packets concerning commits or aborts that occur between checkpoints are held back** and because any writes to the standby are buffered until a checkpoint completes, in the case of a failure any commits or aborts that are not copied over to the standby are never seen by the client either. Therefore when the standby takes over, its state is consistent with the state seen by the client. In other words, **the client never sees inconsistent state.**
- There is always a **crash consistent disk image.**

Problems with Remus

- **Remus checkpoints *all* of the VM's memory**. Because DBMS systems use large amounts of memory relative to other workloads, **this leads to significant overhead**.
- **Network buffering** (holding back packets between checkpoints) becomes the **source of significant network latency**. But DBMS have their own notions of consistency and therefore not all outgoing packets need to be buffered before by Remus.

Essentially, Remus is designed for any kind of application and therefore is not as well optimized as it could be for any particular type of application. This is why **our authors set out to create RemusDB: Remus optimized for DBMS applications**.

RemusDB: Memory Optimizations

- Compressing Checkpoints
- Disk Tracking
- Memory deprotection

Compressing Checkpoints

DBMSs often make small changes to large pages of memory. Therefore **a lot of the data that Remus would checkpoint is redundant.**

Therefore RemusDB tries to send only the changes to the data.

This is accomplished using a **LRU cache** and the **following algorithm** to be applied to every page of memory that is sent to the standby.

- Check page against the cache.
- If an older version of it is in the cache XOR the page against its older version and send that.
- Otherwise send the page as is and add the document to the cache.

Disk Tracking: What Remus Does

In normal Remus:

- All of the page table entries in memory are initially set to read-only producing a trap when something is modified
- Once a **modification of a page** is requested the trap handler verifies the write is okay and then **updates a bitmap of dirty pages.**
- **All dirty pages are sent on each checkpoint.**
- **After** each **checkpoint** this **bitmap is cleared.**

Disk Tracking: What RemusDB Does

- Normally a read from disk into a memory page would cause that page to be marked as dirty.
- **In RemusDB, a read from disk does not cause a page to be marked as dirty,** only modification of a page in memory causes it to become dirty.
- Therefore **pages filled with unmodified data read from disk are not sent with the checkpoint**
- Instead, **RemusDB adds an annotation** to the replication stream **indicating where to read the data from.**
- To keep track of all of this, RemusDB creates a read tracking list.
- RemusDB also keeps a set of references from these pages to the associated sectors on disk. **If the VM writes to any of these sectors, the corresponding page is removed from the read tracking list and the page becomes dirty.**

Memory Deprotection

The authors considered a third idea: **allow the administrator to specify certain regions of memory to remain unprotected**.

These areas of memory would not be replicated during checkpoints but instead would be tracked so that a *failover handler* could either regenerate them or destroy references to them in the event of a failure.

Problem:
Such tracking causes CPU overhead for RemusDB. The only data structure they could identify which justified the CPU overhead and complexity for the user was reads from disk. But this was already handled more efficiently and transparently by read tracking. Therefore **memory deprotection did not end up as part of RemusDB**

RemusDB: Memory Optimization Summary

- Compressing Checkpoints
  - Less redundant data is sent at each checkpoint
- Disk Tracking
  - Simple reads from disk are not sent in checkpoints
  - Rather the standby is simply told where to read the data from
- Memory deprotection
  - Not used

RemusDB: Network Optimization

- Commit Protection

Commit Protection: The Problem

- Normally **Remus buffers all outgoing packets**
- **This is overconservative because DBMSs have their own notions of consistency.**
  - Only things like commit and abort really need to be protected
- Therefore the group decided to **allow the DBMS to determine which packets should be buffered.**

Commit Protection: The Solution

Allow the DBMS to determine which packets should be buffered.

How RemusDB allows this:

Gives the DBMS the ability to switch between a protected mode and an unprotected mode for any connection. Protected connections would buffer their packets as would normally happen with Remus, unprotected connections act like connections on an unprotected VM.

- Protected and unprotected Mode implemented by adding a new setsockopt() option to Linux
- **The DBMS system itself needs a small modification that causes it to trigger this option on connections when appropriate.**

Commit Protection: How it Should be Used

1. If the server receives a commit or abort it switches the connection to protected mode.
2. Performs necessary actions
3. ‘Send’ the message in protected mode
4. Once the message has been sent, switch back to unprotected mode

Therefore **a commit or abort must be propagated to the standby before the client can see it**, but **less important client-server communication** is not buffered. This will result in a **decrease in network latency** over normal Remus.

An Issue with Commit Protection

TCP connection state can be lost on failover. Therefore the unbuffered packets advance TCP sequence counters which can result in the connection stalling until it times out.

They did not address this problem yet as it only affected a small set of connections and those connections were always recovered after the initial connection timed out.

RemusDB: Network Optimization Summary

- Commit Protection
  - Only messages tied to DBMS-specific consistency are buffered.

Experimental Evaluation

Two Benchmarks

- TPC-C

1. An on-line transaction processing (OLTP) benchmark.
2. Simulates a complete computing environment where a population of users executes transactions against a database

- TPC-H

1. A decision support benchmark
2. Examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. (a read intensive workload)

Experimental Evaluation

Three optimizations of RemusDB

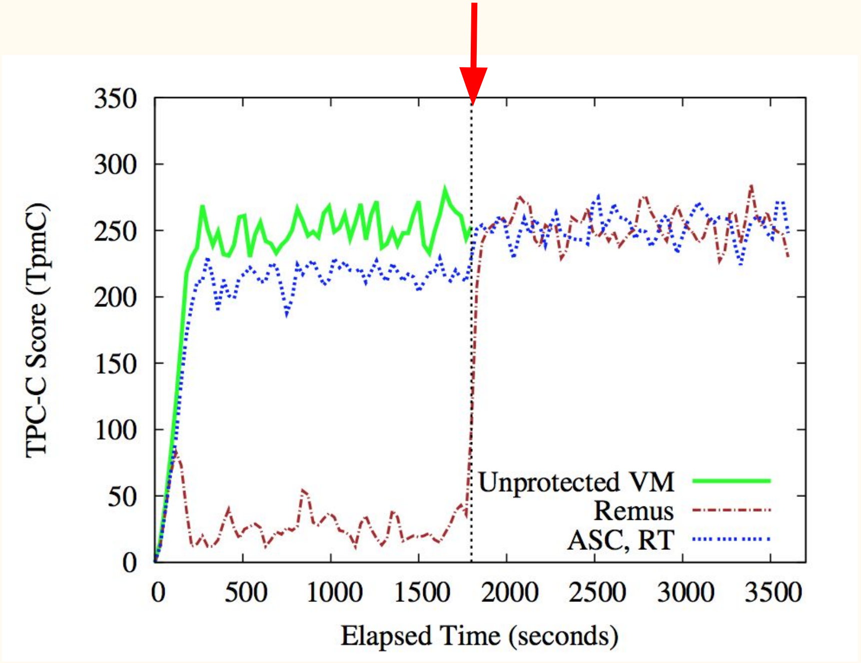
- ASC
  Asynchronous checkpoint compression -- sending less data
- RT
  Disk read tracking -- protecting less memory
- CP
  Commit Protection -- allow DBMS to decide which packets to hold or send to client

Experimental Evaluation

Behaviour of RemusDB During Failover

- VM at the backup recovers with ≤ 3 seconds of downtime and continues execution.
- The performance of RemusDB with ASC and RT is much better than Remus
- After failure, performance rises sharply because VM is not protected

A failure is simulated by cutting power

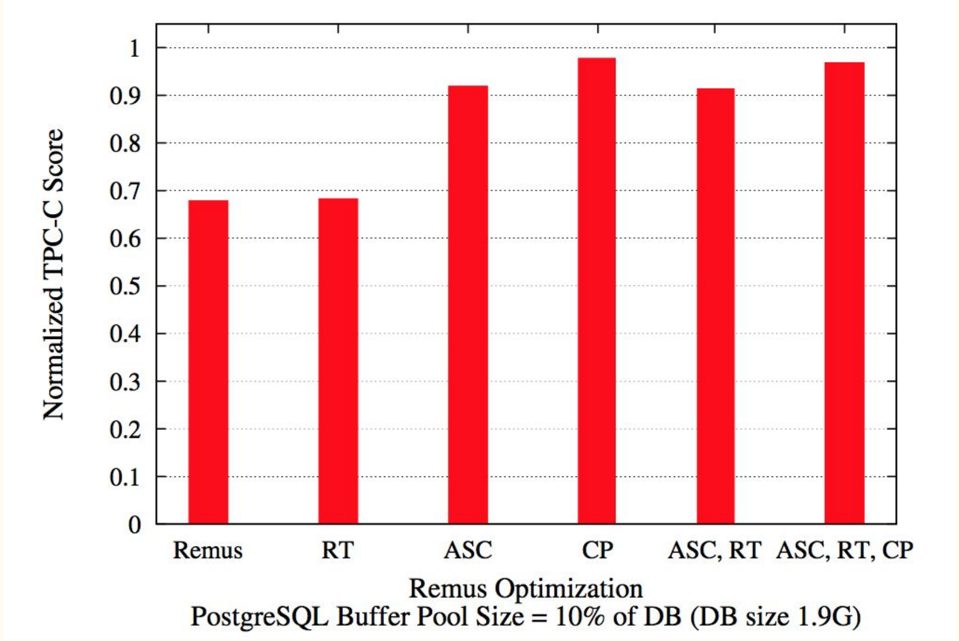


TPC-C Failover(Postgres)

Experimental Evaluation

Overhead During Normal Operation (TPC-C)

- Remus protection for database systems comes at a very high cost.
- The RT optimization provides very little performance benefit because TPC-C has a small working set and dirties many of the pages that it reads.
- ASC and CP provide significant performance gains because both of them help reduce network latency to which TPC-C is particularly sensitive

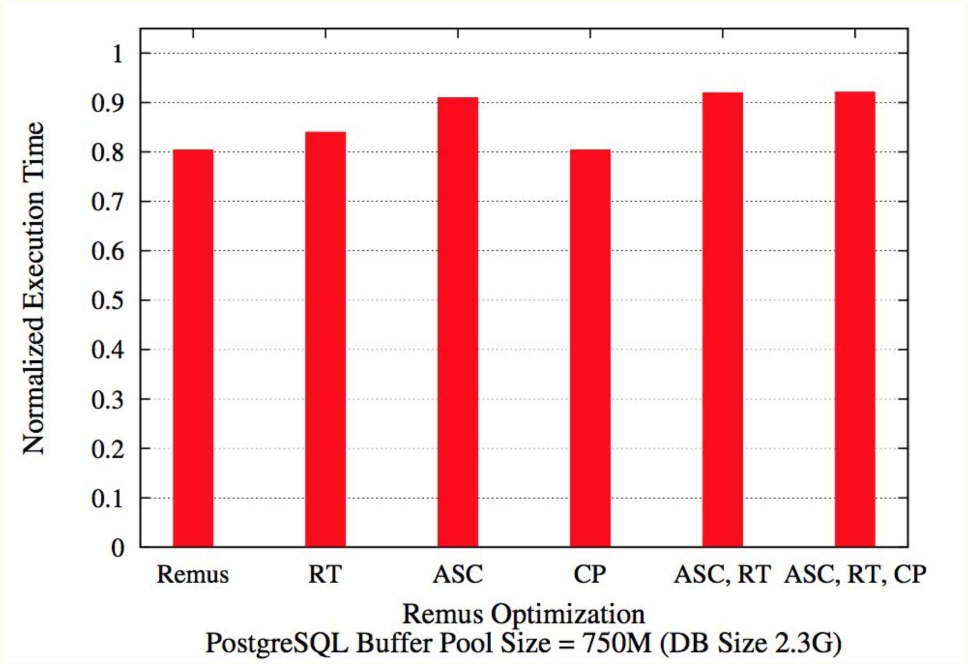


TPC-C Overhead(Postgres)

Experimental Evaluation

Overhead During Normal Operation (TPC-H)

- No gain with CP because it is insensitive to network latency
- Gain the most benefit from memory optimizations alone (ASC and RT), because this TPC-H is a read intensive workload

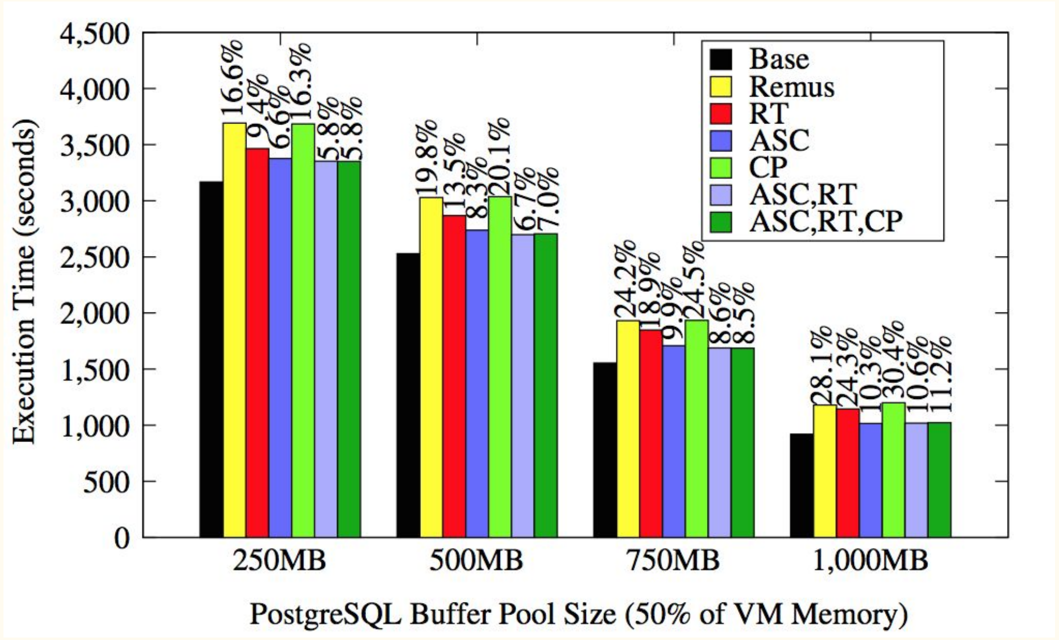


TPC-H Overhead(Postgres)

Experimental Evaluation

Effects of DB Buffer Pool Size

- Study the effect of varying the size of the database buffer pool(the database size is same) on memory optimizations
- Memory optimizations(ASC,RT) offer significant performance gains
- Benefit of RT decreases with increasing buffer pool size. Because smaller buffer size result a lot disk reads, which makes RT more useful

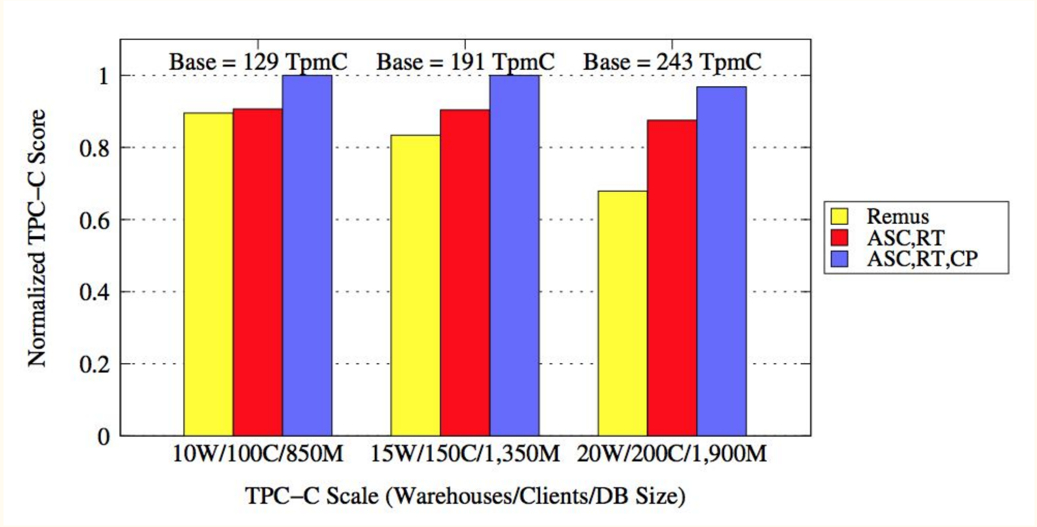


Question: Why use TPC-H workload for this experiment? $\longrightarrow$ TPC-H workload

Experimental Evaluation

Effects of Database Size(TPC-C)

- Overhead of unoptimized Remus increases considerably, going from 10% to 32%.
- RemusDB with memory optimizations (ASC, RT) incurs an overhead of 9%, 10% and 12%
- RemusDB with memory and network optimizations (ASC, RT, CP) provides the best performance at all scales, only a 3% overhead in the worst case



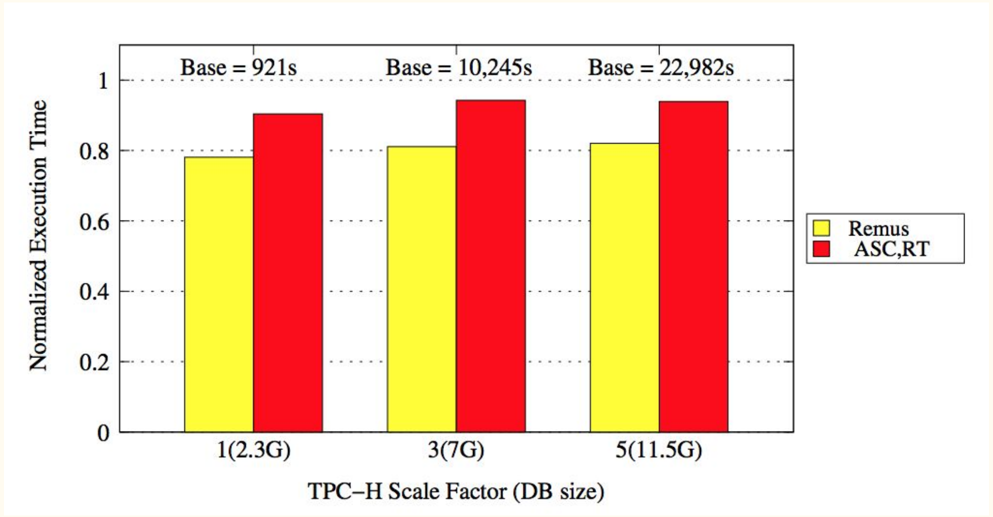
TPC-C workload

Experimental Evaluation

Effects of Database Size(TPC-H)

- Unoptimized Remus incurs an overhead of 22%, 19%, and 18% for different database size

- RemusDB with memory optimizations has an overhead of 10% for scale factor 1 and an overhead of 6% for both scale factors 3 and 5 – showing much better scalability.



TPC-H workload

Conclusion

- RemusDB provides active-standby HA and relies on VM checkpointing to propagate state changes from the primary server to the backup server based on Remus VM.

- Identified two causes for performance overhead of database under Remus
  1. Amount of state that needs to be transferred from primary to backup because database systems use memory intensively
  2. Database workloads can be sensitive to network latency.

- Several optimization help RemusDB impose little performance overhead
  1. Asynchronous checkpoint compression -- ASC
  2. Disk read tracking -- RT
  3. Commit Protection -- CP

References

- Minhas, U. F., et. al (2011) RemusDB: Transparent High Availability for Database Systems
- Cully, B., et. al Remus: High Availability via Asynchronous Virtual Machine Replication
- White paper: Oracle Real Application Clusters
- Implementing High Availability Cluster Multi-Processing (HACMP) Cookbook
- High Availability and Disaster Recovery Options for DB2 for Linux, UNIX, and Windows
- https://dev.mysql.com/doc/refman/5.7/en/replication-howto-masterstatus.html
- http://www.tpc.org/tpcc/
- http://www.tpc.org/tpch/