NCAR | National Center for
UCAR | Atmospheric Research

Statistic Multiplexed Computing (SMC) – The Neglected Path to Unlimited Application Scalability

Justin Y. Shi | shi@temple.edu

April 2, 2013

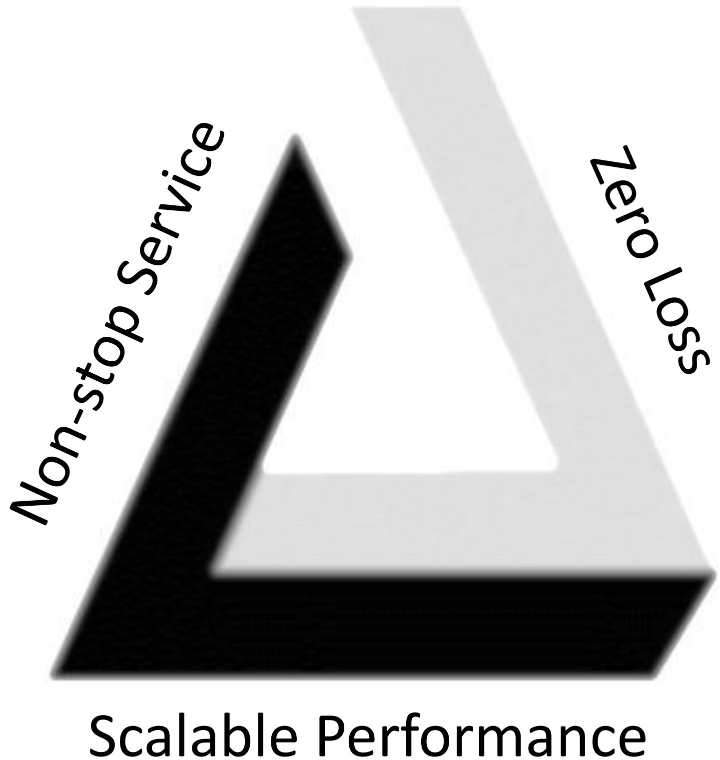
Software Engineering Assembly
NCAR | Boulder | CO

TEMPLE UNIVERSITY

Myths in Computing Science

- To get performance, reliability must be sacrificed
- To gain reliability, performance must be sacrificed
- It is very difficult, if not impossible, to eliminate single-point failures

Impossible Triangle



A true solution for **ANY** is the solution to **ALL**.

Agenda

- The First Principle of Extreme Scale Applications
- The “Smoking Gun”: Why explicit parallel programs are hard to scale?
- The “refresh button” and elimination of single-point failure
- SMCA for Compute Intensive Applications
  - Why Amdahl’s and Gustafson’s Laws are not useful
  - Timing Model: A Software Engineer’s slipstick
  - A Blueprint for Exascale Processors
  - The Second principle of extreme scale HPC application engineering
- SMCA for Data Intensive Applications
  - CAP Theorem and Two Curious Assumptions
  - A Blueprint for Internet-sized Data Intensive Processor
  - Inductive computational results
- Summary and Q/A

Extreme Scale Applications

- Exascale Computing
- Big Data Processing

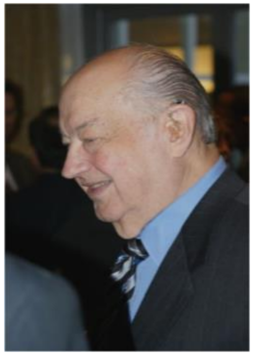
The First Principle in Extreme Scale Software Engineering

Ability to Harness Volatile Resources

Theoretical Limits

- Perfect data communication is **impossible** if the probability of component failure is greater than zero [Lynch 1993].

- Statistic Multiplexing or packet switching [Baran 1960] enabled harnessing resource volatility for data networks.



Architecture Dichotomy

- For data communication architectures, adding routers and switches enhances performance and reliability at the same time. **Scalability has no limit**.
- For distributed (and parallel) computer architectures, adding nodes can either enhance performance or reliability, not both. **Scalability is challenged.**
- Observation: All applications are built using scalable data networks. **What went wrong?**

In Search for the Weakest Link

- All distributed and parallel application programming interfaces (API) assume **reliable application-level communication.**
- Operating system **hands off** all communication tasks to the protocol stack.
- Communication stack can be characterized in **7 layers** (OSI).(regardless actual implementations)

The Imperfect (OSI) Layers

	Data unit	Layer	Function
Host layers	Data	7. Application	Network process to application
		6. Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data
		5. Session	Interhost communication, managing sessions between applications
Media layers	Segments	4. Transport	End-to-end connections, reliability and flow control
	Packet/Datagram	3. Network	Path determination and logical addressing
	Frame	2. Data link	Physical addressing
	Bit	1. Physical	Media, signal and binary transmission

Why Explicit Parallelism is Bad?

- The protocol stack is processed on the **host computers**. The media layers are built-in the **network adaptor**.
- The network adaptor make sure that the data plane is intact regardless component failures.
- The host layers are processed by the host processor. **Any transient failure on the path of packet->application-level processing can hang the entire application**.
- Bigger applications deploy more processing nodes. **The probability of failure increases proportionally as we up scale the application.**

The “Smoking Gun”

	Data unit	Layer	Function	API Vulnerability
Host layers	Data	7. Application	Network process to application	
		6. Presentation	Data representation, encryption and decryption, convert machine dependent data to machine independent data	
		5. Session	Interhost communication, managing sessions between applications	
Media layers	Segments	4. Transport	End-to-end connections, reliability and flow control	
	Packet/Datagram	3. Network	Path determination and logical addressing	
	Frame	2. Data link	Physical addressing	
	Bit	1. Physical	Media, signal and binary transmission	

Effects of Cumulative Transient Errors

- Transient failures are the primary reasons for application and storage failures.
- The number of failures cumulates as we up scale the complexities of software and hardware.

Fact Check: The MTBF for a multiprocessor of 1024 nodes is shrinking to less than 60 minutes [Gibson2007]. http://www.pdsi-scidac.org/

Between "Refresh button" and Single-point Failures

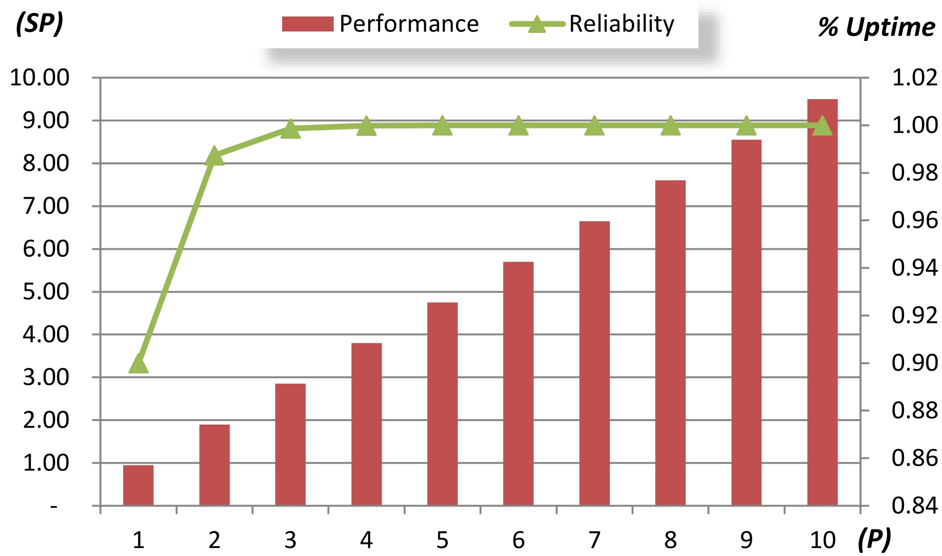
- **Question**: What would be your Internet experience, if the refresh button is removed?
- What really happens when you push the refresh button?

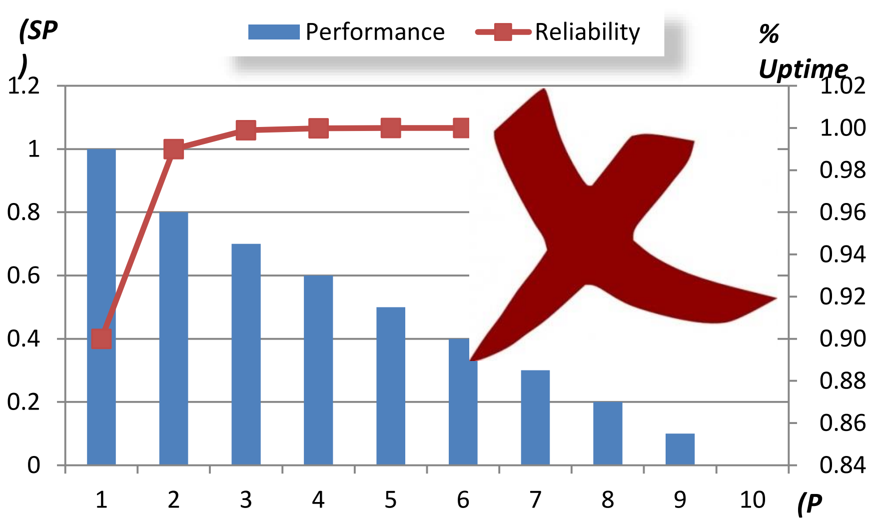
It eliminates ALL single-point failures.

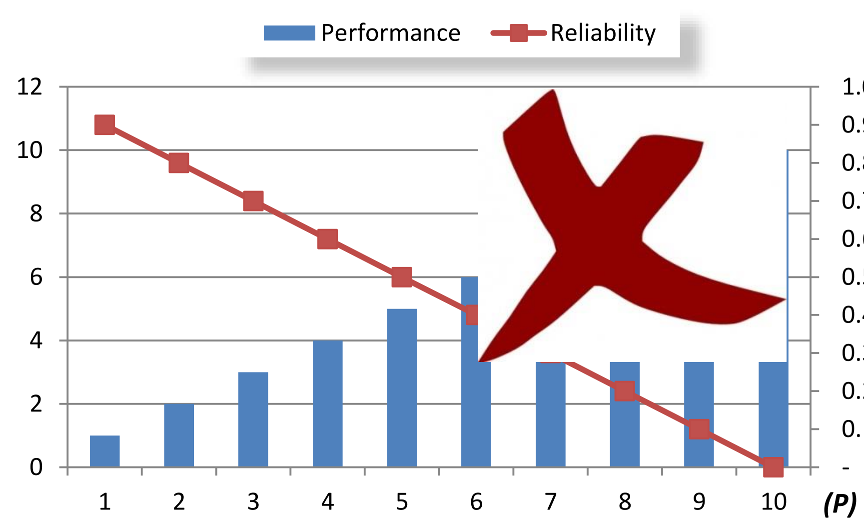
Lessons from History

- The Internet today is not a result of incremental improvements over circuit-switching networks.
- A paradigm shift, packet switching, was necessary.
- **Analogy**: Explicit parallel paradigms are building one off “circuit-switching” networks.

Architecture Objective







Architecture Engineering

- Statistic Multiplexed Computing Architectures (SMCA): there Must be **multiple paths** for processing the same task (definition of mission apps?)
- Structural support for **temporal and spatial redundancies** to exploit all possible volatile hardware/software components.
- Must **eliminate the reliable communication assumption in ALL APIs** (timeout discipline)

Devils are in the Details

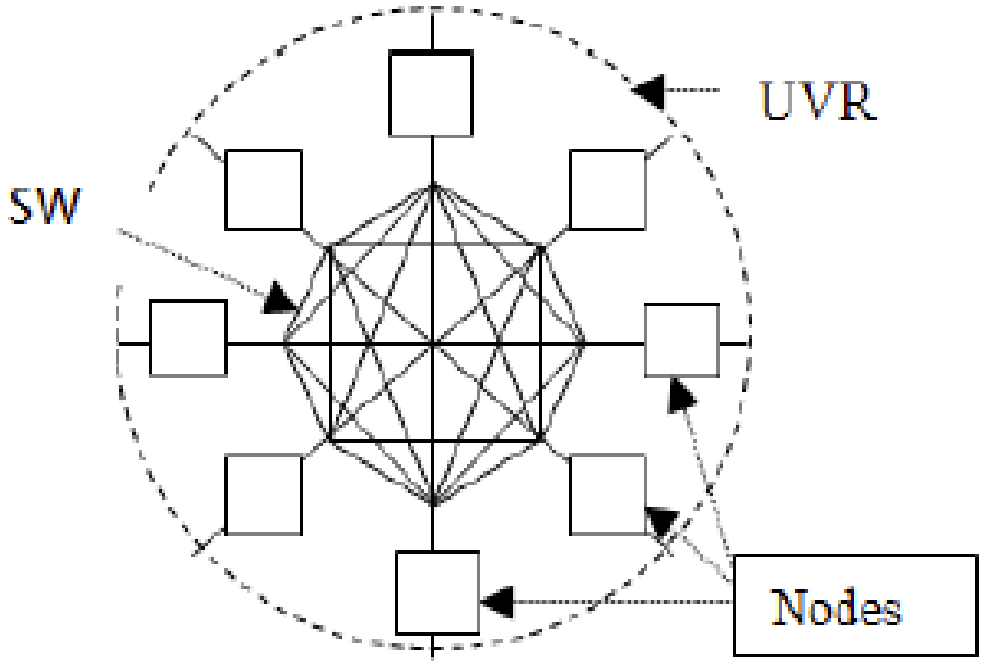
- All APIs must include SMCA semantics: re-transmission on timeout with uniqueness check (**what do we teach students about timeout**?)
- Infrastructure must support SMCA semantics: store-and-forward (**remember the 55yr old debate**?)
- What about the impossibilities in **CAP Theorem**?

2 Application Types

- **Compute Intensive (CI)**: Not every state change needs to be saved. Example: HPC apps.
- **Data Intensive (DI):** Every data state change needs to be recorded permanently. Example: Transaction Processing, Data Storage.

CI: A Blueprint for [Exascale HPC]

Tuple-Switching Network (implicit data parallel)

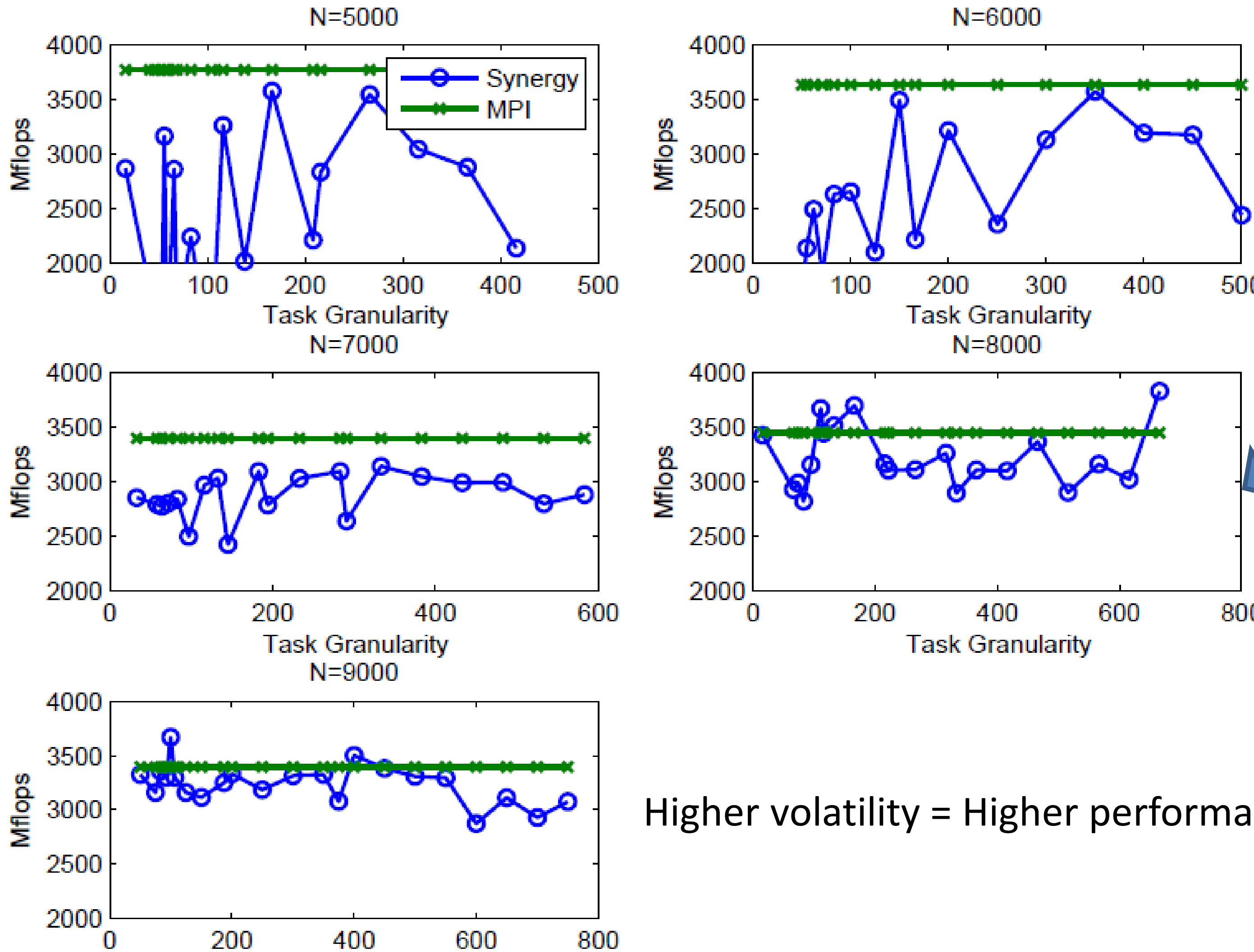


Automatic Data Reduction Machine

How It Works

- Applications are decomposed into data-parallel segments.
- A Master generates working tuples for processing.
- Workers are automatically generated on multiple nodes to process different tuples.
- The Master collects the results when done.
- Only Master needs checkpoints (***O(1)***). Workers (***O(p)***) are automatically protected by SMCA tuple re-transmission mechanism.
- Performance is tunable by changing granularity (umesh?).

Computational Results



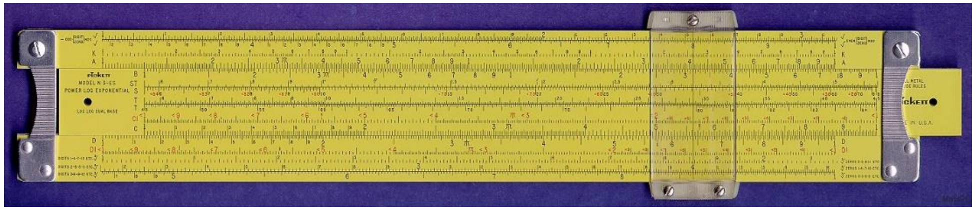
Higher volatility = Higher performance

The Second Principle in HPC Software Engineering

- Parallel programs must be tunable after compilation.
- Otherwise, it is impossible to expect high performance from the same code in different processing environments.
- Explicit parallel and functional programming paradigms have violated the second principle.

Why the Laws are Not Useful

- Amdahl's and Gustafson's Laws: single measure of parallel v.s. sequential percentage.
- They are related.
- **Question**: How can you predict application scalability without quantifying communication?
- Timing analysis (software engineer's slipstick) shows each application is only limited to scale to whatever the processing architecture can support.



DI: The CAP Confusion

- Eric Brewer proposed the CAP conjecture in 2000: One can only expect at most two of the three desirable properties for a (data intensive) web service: data **c**onsistency, **a**vailability and (network) **p**artition tolerance.
- In 2002, Gibert and Lynch published an informal proof of CAP. It is now called the CAP Theorem.

CA, CP or AP?

- Data consistency was the first to be sacrificed. Google led the charge: GFS.
- NoSQL, Casandra, StreamDB, MongoDB, etc.

- **Question**: Can consistency relaxed data sources be used for mission critical apps?

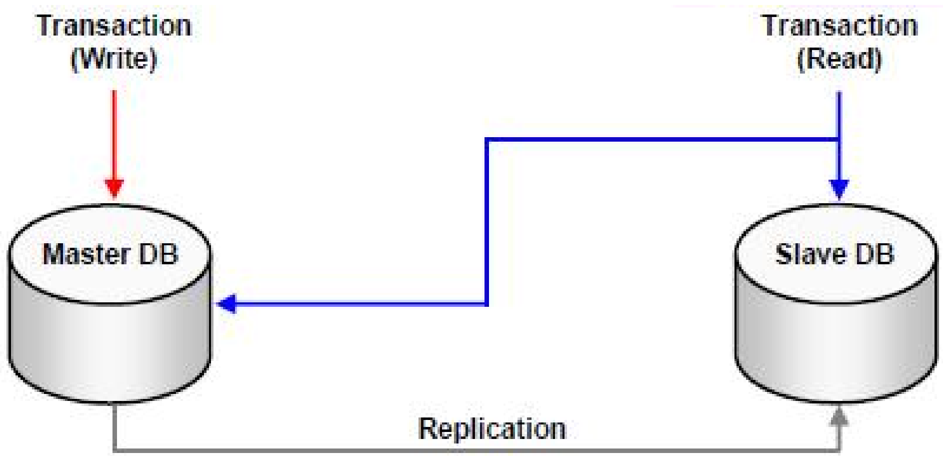
Single Difficulty: Replication

Current industry standards:

- Synchronous with 2PC protocol
- Asynchronous

The Story of 1+1 < 1

- Synchronous Replication with **2PC**:



- 2 Servers deliver less than 1 server's performance.
- 2 servers deliver less than 1 server's availability.

The Story of 1+1 (cont)

- Asynchronous Replication:

- 2 servers deliver less than 1 server's performance.
- 2 servers delivers less than 1 server's reliability.

2 Curious Assumptions in CAP

- Arbitrary message loss
- Atomic replication with 2PC

Why?

Arbitrary Message Loss

- Transaction processing API assume reliable transaction processing -> every transaction is **only transmitted once**. Thus in theory, transaction loss cannot be prevented (like UDP).

- **Observation**: Only statistic multiplexing transaction can eliminate arbitrary losses.

Synchronous Replication with 2PC

- Failure in **any** replication target will cause the transaction to rollback.

- **Observation**: **Every** target server's state is semantically acceptable to all apps. Why throw the baby out with the bath water?

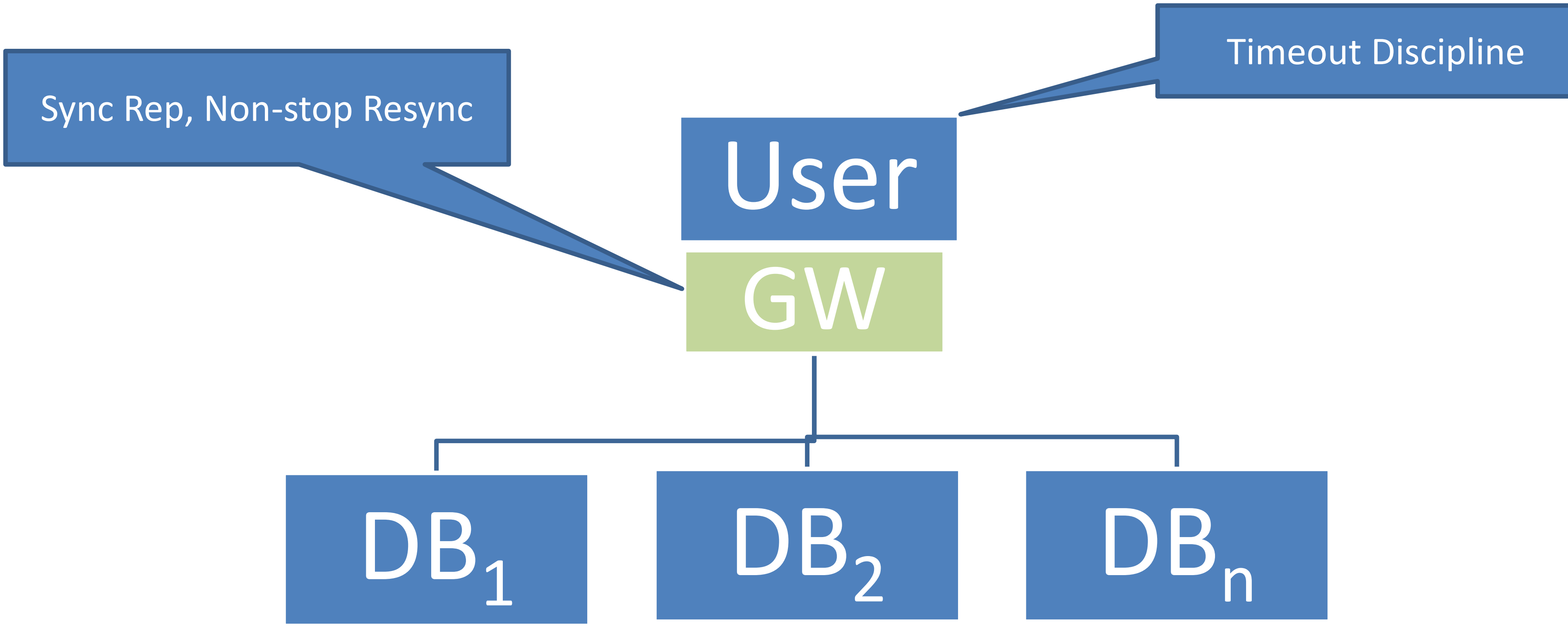
Necessary Condition

- For trustworthy data intensive applications, data consistency is the necessary condition.
- The same condition is necessary for statistic multiplexed DI computing architecture.

Reality Check

- Database engines all serve as the concurrent update conflict resolver while replicating (federated then serialized).
- **Question**: Since either the primary or the secondary is equally likely to be responsible for data inconsistencies, why artificially appoint a "primary"?

Statistic Multiplexing DI Architecture (counter intuitive)



(DB^x Architecture)

Single-point Failure?

- No problem, if all **clients** are timeout disciplined (automated re-fresh button)

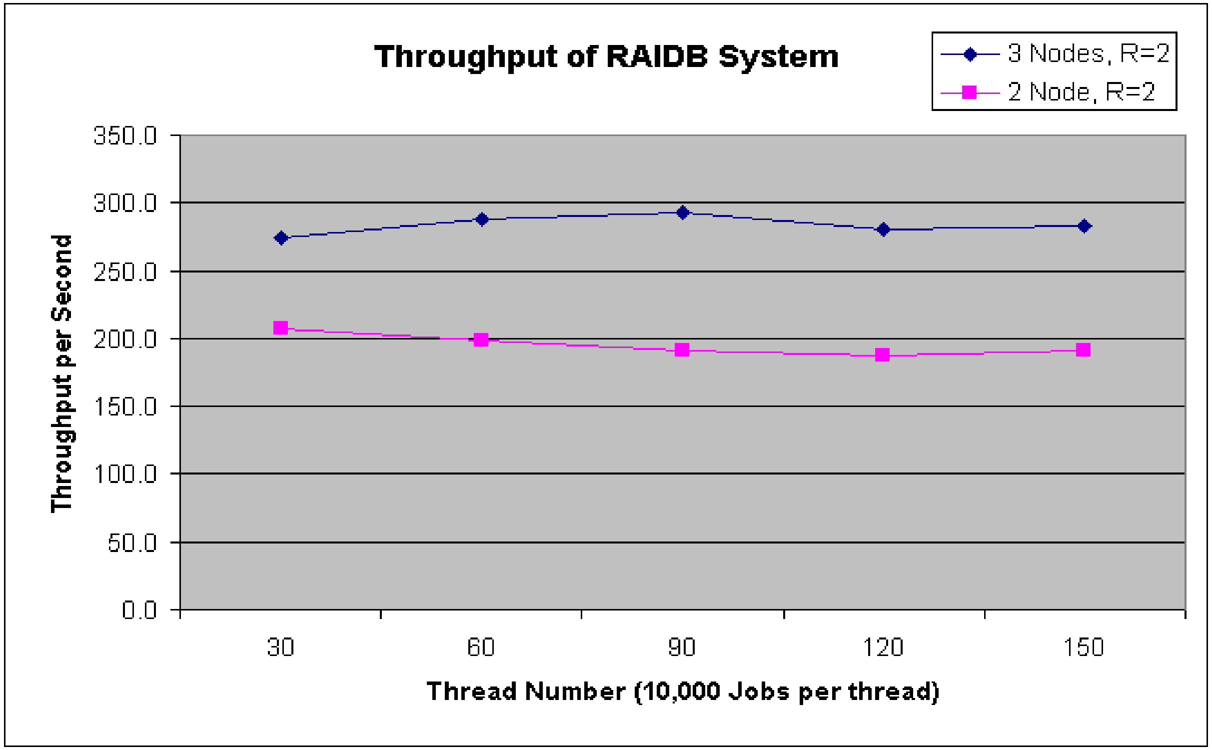
Unlimited DI Performance?

- Storage overhead is the ultimate performance bottleneck for all DI applications.
- Data partitioning is the proven load distribution method (with **a catch**: every new server is a new single-point failure)
- Solution: **P >> R**
- You can add servers **indefinitely** by keeping a small **R**.

Inductive Experiments

Thread Number	Case A: 3 Servers		Case B: 2 Servers		Throughput Speedup
	Elapse Time,s	Throughput	Elapse Time,s	Throughput	
30	1095.2	273.9	1450.0	206.9	1.324
60	2086.4	287.6	3023.7	198.4	1.449
90	3075.2	292.7	4710.5	191.1	1.532
120	4280.7	280.3	6385.9	187.9	1.492
150	5291.5	283.5	7861.2	190.8	1.486

Average throughput speedup **1.456**



Broad Impacts

- Exascale Computing
- Internet-sized Big Data Processing
- Internet-sized Storage Networks
- Lossless Transaction Processing Networks
- Lossless Service Oriented Architectures (SOA)
- Mission Critical Applications
- … and the way we teach CS

Acknowledgements

- Reported CI architecture research is supported in part by National Science Foundation (MRI)
- DI architecture research is supported in part by Ben Franklin Technology Partners and private investors to Parallel Computers Technology Inc.

Request for Collaborators

- Looking for collaborators for the upcoming SC13 (Denver, Nov) research exhibit
  - Compute intensive apps (collaboration for a demo app)
  - Data intensive apps (collaboration on the development of P2PHDFS project)
- Contact: shi@temple.edu

Q&A

Speaker Bio

Justin Y. Shi is an Associate Professor and Associate Chairman of Computer and Information Sciences Department of Temple University. He earned his B.S. in Computer Engineering from Shanghai Jiaotong University in 1977, M.S. and Ph.D. in Computer Science from the University of Pennsylvania in 1983 and 1984 respectively. He was elected Chairman for Computer and Information Sciences Department from 2007-2009. He is also the founder and Chairman of Parallel Computers Technology Inc., an independent research and development company in King of Prussia, Pennsylvania. He has consulted for the Department of Homeland Security and the Department of Human Services of Philadelphia. His research has been supported by the National Science Foundation, National Institute of Health, IBM T.J. Watson Research Center, Microsoft, Amazon.com and other private companies.