Exhibit: Experiences for Extreme Scale Computing

Justin Y. Shi | shi@temple.edu

Center for Network Computing
Motivation

Extreme Scale **Data Intensive** Parallel Computing
21\textsuperscript{st} Century Software Engineering Challenges

• How to gain performance and reliability at the same time when upscaling the processing infrastructure?
• How to eliminate all single point failures?
• How to deliver non-stop services?
A solution for **ANY** is the solution to **ALL**.
Theoretical Extremes

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

• Statistic Multiplexing or packet switching [Baran 1960] has delivered **infinitely scalable** networks nearing the perfect states harnessing massive volatilities in extreme scales.
Architecture Dichotomy

• For data communication architectures, adding routers and switches enhances performance and reliability at the same time. “Diminishing returns” applies but **Scalability has no limit.**

• For distributed (and parallel) computer architectures, adding nodes can either enhance performance or reliability, not both. “Diminishing returns” applies and **Scalability is capped.**

• All computer applications rely on scalable data networks. **What went wrong?**
Distributed Computing Fallacies (Peter Deutsch’94, James Gosling’97, Blog by Arnon Rotem-Gal-Oz’2012)

1. **Network is reliable**: Virtual Circuit concept.
2. **Latency is zero**: Amdahl’s and Gustafson’s Laws.
3. **Bandwidth is infinite**: explicit-parallel.
4. **Network is secure**: Well-know ports for web services.
5. **Topology does not change**: Explicit-parallel
6. **There is one admin**: Perimeter-only security.
7. **Transport cost is zero**: Amdahl’s and Gustafson’s Laws.
8. **Network is homogeneous**: explicit-parallel.
New Fallacies

1. **Cloud applications are more reliable:** They are not. They are more likely to crash due to resource sharing.

2. **Single Truth Model is sufficient** for identity resolution: It is not. “Data cleansing” before analytics erase digital data evidences.

Broad practice of fallacies also inspire great conjectures:

1. **CAP:** Consistency, Availability and Partition tolerance probably can only be satisfied partially.

2. **Single point failures** probably can never be all eliminated

3. **True scalability** is probably impossible (gaining performance and reliability at the same time as we upscale processing infrastructure).
“Half a Truth is Often a Great Lie.”
Ben Franklin

• The **Half Truth**: Virtual Circuit is an order preserved reliable data communication channel.

• The **Lie** exposed: Any transient software or hardware failure can corrupt the single thread marshaling protocol layer. The probability of corruption increases exponentially as we increase the number of communicating programs.
<table>
<thead>
<tr>
<th>Host layers</th>
<th>Data unit</th>
<th>Layer</th>
<th>Function</th>
<th>Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data</td>
<td>7. Application</td>
<td>Network process to application</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6. Presentation</td>
<td>Data representation, encryption and decryption, convert machine dependent data to machine independent data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Session</td>
<td>Interhost communication, managing sessions between applications</td>
<td></td>
</tr>
<tr>
<td>Media layers</td>
<td>Segments</td>
<td>4. Transport</td>
<td>End-to-end connections, reliability and flow control</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Packet/Datagram</td>
<td>3. Network</td>
<td>Path determination and logical addressing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame</td>
<td>2. Data link</td>
<td>Physical addressing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bit</td>
<td>1. Physical</td>
<td>Media, signal and binary transmission</td>
<td></td>
</tr>
</tbody>
</table>
Failure Projection

Meantime to interrupt is already less than 60 minutes at 100k socket level.

Credit: Garth Gibson, 2007
A Bridge to Nowhere: Check pointing

- The growing probability of failure will drive more CPR per parallel application
- More CPR -> Less utility (loose half machine quickly)
- Pressure -> Disks
- Disk failure also grows in scale -> scary numbers ...
- Needs a paradigm shift [Garth Gibson 2007]
Proposed Paradigm Shift: Explicit Parallel -> Implicit Parallel

Base Case

Explicit Parallel

Task1
Processor1

Single Task

Multiple Tasks
(Better Performance
But Less Reliability)

Implicit Parallel

Task1
Processor1

Single Task

Multiple Tasks
(Better Performance
And Better Reliability)

Induction: Higher Performance and More Reliability

Scalability has NO upper bound
Unlimited Scalability and “Diminishing Returns”

• “Diminishing returns” refers to the fact that given problem size, the resource efficiencies diminishes as we add more resources.

• Unlimited scalability refers to our desire to deliver incrementally better performance and reliability as we add more resources into solving a growing problem.

• Both can be TRUE.
Missing Discipline

- **Fact**: Only retransmission discipline can eliminate ALL single-point failures, if components are statistic multiplexed.

- **Implication**: Adding retransmission discipline means a **paradigm shift**: from circuit-switching thinking -> packet-switching thinking.

**Reality Check**: What if browsers do not have the refresh button?
SMC
(Statistic Multiplexed Computing)
For
Compute Intensive Applications (CI)
Case Study: **Tuple Switching Network**

- **Application**: Parallel Matrix Multiplication
- **Platforms**: MPI vs. Jenergy (Java + Zookeeper + Cassandra tuple space implementation)
- **Architecture Difference**: Explicit-parallel (MPI without checkpoints) vs. Implicit-data (tuple space) parallel with real time “stable storage” (complete automatic tuple and process quorum backup and recovery)
- **Application Difference**: Fixed N/P partition vs. variable partition (< N/P units (granularity) assigned to each worker)
- **Zero CPR**: statistic multiplexing master programs in addition to Workers (*limitations apply*)
- **Data Protection**: Every tuple in Jenergy is replicated to 3 different nodes using a quorum synchronization protocol (Cassandra).
# Zero-CPR Application Types

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic</td>
<td><strong>Deterministic</strong></td>
</tr>
<tr>
<td>Deterministic</td>
<td>Supported. Examples: Linear and nonlinear solvers, FD, Molecular simulation, etc.</td>
</tr>
<tr>
<td>Non-Deterministic</td>
<td>Supported. Example: Parallel search, unstable parallel sort, etc.</td>
</tr>
<tr>
<td>Non-Deterministic</td>
<td><strong>Non-Deterministic</strong></td>
</tr>
<tr>
<td>Non-Deterministic</td>
<td>Not Supported. Example: Parallel random search, random date-time based reduction</td>
</tr>
<tr>
<td>Non-Deterministic</td>
<td>Supported. Example: Monte Carlo simulations, etc.</td>
</tr>
</tbody>
</table>
First Performance Surprise

Granularity Tuning (Jenergy-TCP vs MPI-TCP)
Matrix Multiply: N=6000 | P=16 | MaxG=N/P

- Lower is Better
- Max: N/P

Elapsed Time (s) vs Granularity size

Graph showing performance comparison between Jenergy-TCP and MPI-TCP for matrix multiply with N=6000, P=16, and MaxG=N/P. The graph indicates that Jenergy-TCP (Quorum) generally outperforms MPI-TCP in terms of elapsed time, with lower values indicating better performance.
Discussions

• The largest tuple is $N^2N^*8$ bytes $= 288$MB (matrix B)
• Each task tuple is $N^*G^*8$ bytes, $G=$granularity, between $2.4$MB – $18$ MB.
• Jenergy performance is tuned to seek “Termination Time Equilibrium”. MPI task granularity is fixed $N/P=375$.
• If we use $P=1$ as the base case, the reported performance implies, inductively, there is no limit on this application scalability, and under the rule of “diminishing returns”.
• In other words, the deliverable performance is decided by the application size and the aggregate of exploitable processing powers. Note that application partitioning factor plays a critical role in the exploitable processing powers.
Why?

• Processing elements are inherently volatile even in dedicated environments

• Fixed N/P partition (or granularity) hurts deliverable performance since the optimal granularity is always smaller in heterogeneous environments

• Implicit data parallel applications allows dynamic tuning of partition factor to compensate imbalances. Thus it out-performs fixed partitioned bear-metal MPI applications (even with disk replicated runtime tuples)
Load Balancing is the New Performance
SMC Requirements

• A well-defined unit of transmission
• Structural support for temporal and spatial redundancies to exploit all possible volatile hardware/software components.
• Application timeouts must be disciplined in order to eliminate ALL single-point failures

Problem:
Incompatible with explicit parallel programs.
SMC Wrapper: Rejuvenate Legacy Codes

- Bigger and better MPI, OpenMP and CUDA programs make bigger and better SMC applications
- Idea: 2-Tier Architecture (like the Internet)
2-Tier Extreme Scale SMC

Tuple Space

Wrapper: SMC Worker
Wrapper: SMC Worker

SMC Master
SMC Master

MPI Workers

4 x 40Gb/s

500M-500GB/s
Case Study

• Application: Parallel Matrix Multiplication
• Platforms: SMC (Synergy) + MPI
• Benchmark Environment: owlsnest.hpc.temple.edu
• HPC Batch System: PBS
2nd Performance Surprise (SMC Wrapped MPI)

Questions and Software Requests: shi@temple.edu
Seeking the Peak Performance: A 16 Core Test

4 Different Config. of 16 Core Tests

- Best Wrapper Performance
- Best MPI Performance

![Bar chart showing performance metrics for 2x8, 4x4, 8x2, and 16x1 configurations.](chart.png)
Get_Gradularity(&g);
T = ceil(N/g);  // Total tasks
// Scatter
for (i=0; i<(T); i++) {
    Build_Tuple(i, g);
    Put_Tuple("i", &i);
}
// Gather
received = 0;
while (received < (T)) {
    Get_Tuple("*", &x);
    Assemble_Result(x, &received);
}
Put_Tuple("Final", 0);  // All Done

MPI_Init();
MPI_Comm_rank(... &myrank);
if (myrank == 0) {
    Repeat:
    Get_Tuple("*", &x);
    if (x == "Final") {
        Put_Tuple("Final", 0);
    } else {
        Get_Gradularity(&g);
        T = ceil(N/g);  // Total tasks
        // Scatter
        for (i=0; i<(T); i++) {
            Build_Tuple(i, g);
            Put_Tuple("i", &i);
        }
        // Gather
        received = 0;
        while (received < (T)) {
            Get_Tuple("*", &x);
            Assemble_Result(x, &received);
        }
        Put_Tuple("Final", 0);  // All Done
        break;
    }
    goto Repeat;
}
Common_MPI_part();
if (myrank > 0)
    Legacy_MPI_Worker();
if (myrank == 0) {
    Legacy_MPI_Master2(&r);
    Build_Tuple(&x, &r);
    Put_Tuple("r", &y);
    goto Repeat;
}
else {
    // Gather
    received = 0;
    while (received < (T)) {
        Get_Tuple("*", &x);
        Assemble_Result(x, &received);
    }
    Put_Tuple("Final", 0);  // All Done
}
The Next Weakest Link: **Storage**

Need HPC Storage revolution:

- Unlimited Internet-scale storage
- Unlimited Internet-scale data processor
- Unlimited Service-Oriented Architectures
- True mission-critical data intensive applications
Application Type Spectrum

- **Compute Intensive (CI):** Not every state change needs to be saved. Example: HPC apps.
- **Data Intensive (DI):** Every data state change may be mission-critical. Example: Transaction Processing, HPC 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.
• A Worker program is automatically launched on multiple nodes to process different tuples.
• The Master collects the results when done.
• Workers are automatically protected by SMCA tuple re-transmission mechanism. No more CPR.
• Masters are also protected by redundant processes with automatic over-write protection*.
• Performance is tunable by changing partition factor (granularity).

* Non-deterministic input + deterministic output is NOT supported
Effects of Granularity Tuning

Higher volatility = Higher performance

High is better
A Principle for Extreme Scale Application Engineering

• Processing granularity 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 violate the second principle.
Fallacy Affected Laws

• **Amdahl’s** and **Gustafson’s Laws**: single measure of parallel vs. sequential percentage.
• They are mathematically **related**.
• **Fallacy connections**: Zero latency and zero transport cost.
• **Timing analysis** (software engineer’s slipstick) shows each application is only limited to scale to whatever the processing architecture can support.
DI: The **CAP** Challenges

- 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, Gilbert and Lynch published an informal **proof of CAP**. It is now called the **CAP Theorem**.
Why is CAP Interesting?

• Intermediate computation results must be stored on stable storage or risk losing the entire calculation.

• Stable storage without performance and reliability limits is theoretically impossible if not statistic multiplexed.

• Statistic multiplexed data storage must solve the CAP challenge since data replication is NOT avoidable.
• Data consistency is often sacrificed to deliver performance and availability.
• Examples: GFS, NoSQL, Cassandra, StreamDB, MongoDB, etc.

**Question**: Can consistency relaxed data sources be used for mission critical apps like HPC 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 < 1 (cont)

- Asynchronous Replication:
  - 2 servers deliver less than 1 server’s performance.
  - 2 servers delivers less than 1 server’s reliability.
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 server’s state is semantically acceptable to all apps. Why throw the baby out with the bath water?
Why is C Necessary?

• For trustworthy data intensive applications, consistency in all replicated data is the minimal necessary condition.
• Thus at least causal consistency 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 Multiplexed Transaction (SMT)

(DB^x Architecture)
The Mythical Triangle Check

• Non-stop computing is promised by the non-stop resynchronization algorithm (DBx)

• Transaction loss can be statistically multiplexed to approach zero.

• There will be no performance upper bound (next slide).

• Sing-point failures = Zero, since 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 but with a **drawback**: every new partition is a new single-point failure

• Solution: \( P >> R \)

• You can now add servers **indefinitely** by keeping a smaller \( R \).
**Inductive Experiments (SMT)**

<table>
<thead>
<tr>
<th>Thread Number</th>
<th>Case A: 3 Servers</th>
<th>Case B: 2 Servers</th>
<th>Throughput Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Elapse Time, s</td>
<td>Throughput</td>
<td>Elapse Time, s</td>
</tr>
<tr>
<td>30</td>
<td>1095.2</td>
<td>273.9</td>
<td>1450.0</td>
</tr>
<tr>
<td>60</td>
<td>2085.4</td>
<td>287.6</td>
<td>3023.7</td>
</tr>
<tr>
<td>90</td>
<td>3075.2</td>
<td>292.7</td>
<td>4710.5</td>
</tr>
<tr>
<td>120</td>
<td>4280.7</td>
<td>280.3</td>
<td>6385.9</td>
</tr>
<tr>
<td>150</td>
<td>5291.5</td>
<td>283.5</td>
<td>7681.2</td>
</tr>
</tbody>
</table>

Average throughput speedup: 1.456
Introducing ANKA: A Data Intensive Parallel Processing Engine

• Statistic multiplexed <name, value> pair computing engine with integrated storage
• Data replicated consistently at all times
• Can gain performance and reliability at the same time when adding components
• Zero CPR for non-stop services
ANKA <name,value> Pair Computing and Query/Storage Engine
ANKA Node Architecture

1. **Node Manager**: responsible for the physical node’s position in the UVR, worst complexity $O(\lg P)$.

2. **Replication Gateway**: Any request made to the ANKA is first handled by this component. Request arrives in the form of tuples that encapsulate necessary information for Gateway in order to take further actions. Gateway needs to have three necessary conditions required for SMCA. These are dynamic serialization, timeout-retransmission mechanism, and non-stop resynchronization. Dynamic serialization makes sure that there will be no conflicts between storage nodes when replicating data.

3. **Storage Engine**: Each storage engine has certain number of storage units each for specific range. A storage engine has one storage unit for its local gateway and marked as write as well as some other units for to store its predecessor(s) replicas. The job of storage engine is to only service the requests received from either local gateway or remote gateways. All the data received is first kept in memory to be later on flushed into persistent storage components.

4. **Failure Detection, Handling, and Recovery Management**: ANKA has two failure detection mechanisms to detect node failures. The first one works with NodeManager using heartbeat mechanism. Each NodeManager periodically sends heartbeat signals to its predecessor to declare its aliveness. A NodeManager assumes a node fails if it does not receive heartbeat from its successor for a certain period of time. In such a case, a node marks its successor as failed and sends a DECOMMISSION tuple to others. Finally, it replaces its old failed successor with the next alive node in its successors. The second detection mechanism activated in case of failure of message deliveries.
ANKA Performance

Matrix Multiplication: N = 9000

Granularity N/P=82, Saturation Point
Summary

• SMC is probably the only way to mitigate the well-known fallacies out of high performance applications.

• The SMC principles can also be used to eliminate the cyberthreat: the use of well-known ports in secured web services.

• SMC is compatible with any future processor and communication technology advances.
Call for Collaborators

- OLCF Proposal Team, ORNL sponsorship
- Target application designation
- Short-term performance and reliability goals
- Long range research objectives
- Deadline: February 2015
Broader 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 Grant #CNS0958854)
• DI architecture research is supported in part by Ben Franklin Technology Partners and private investors of Parallel Computers Technology Inc.
• NCAR for Yellowstone benchmarks
Track Our Work

SC2013 Exhibit

SC2014 Exhibit