Stateless Parallel Processing Architecture for
Extreme Scale HPC and Auction-Based HPC Clouds

Dissertation Defense Presentation
Moussa Taifi

Dissertation Committee:
Advisor
Dr. Yuan Shi, Temple University
Committee
Dr. Jie Wu, Temple University
Dr. Chiu Tan, Temple University
Dr. Abdallah Khreishah, New Jersey Institute of Technology
External Reviewer:
Dr. Boleslaw K. Szymanski, Rensselaer Polytechnic Institute
Myths in Computing Science

• To get performance, reliability must be sacrificed
• To gain reliability, performance must be sacrificed
• Why do we care?
  – Next slide
What can we do about it?

• The First Principle in Extreme Scale Software Engineering, Shi[13]

Ability to Harness Volatile Resources
Theoretical Boundaries

• 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.
The **BIG SPLIT** in Distributed/Parallel Applications

- 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.
# The BIG SPLIT in Distributed/Parallel Applications

<table>
<thead>
<tr>
<th>Host layers</th>
<th>Media layers</th>
<th>Data unit</th>
<th>Layer</th>
<th>Function</th>
<th>API Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Segments</td>
<td>1. Physical</td>
<td>1. Physical</td>
<td>Media, signal and binary transmission</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Packet/Datagram</td>
<td>2. Data link</td>
<td>2. Data link</td>
<td>Physical addressing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame</td>
<td>3. Network</td>
<td>3. Network</td>
<td>Path determination and logical addressing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Transport</td>
<td>4. Transport</td>
<td>End-to-end connections, reliability and flow control</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Session</td>
<td>5. Session</td>
<td>Interhost communication, managing sessions between applications</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6. Presentation</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>7. Application</td>
<td>7. Application</td>
<td>Network process to application</td>
<td></td>
</tr>
</tbody>
</table>
Why is Explicit Parallelism Vulnerable?

• The Media layers enforce a best-effort service regardless of component failures.
• However, any transient failure on the application-level processing path can hang the entire application.
• Bigger applications deploy more processing nodes. The probability of failure increases proportionally as we up scale the application.
• Imaging that for each packet you sent on the internet, you would setup every router ahead of time.
Patching Explicit Parallelism

• I will focus on the Message passing interface (MPI) in this talk
• Checkpoint-Restart (CPR) is the main mechanism used for disaster recovery in MPI
• The overhead of CPR grows linearly with the number of nodes
• At Exascale, CPR takes longer than the application itself. Patching has limits.
• We need a different method for this problem:
  – Implicit Parallelism with Statistical multiplexed Computing
Statistic Multiplexed Computing

• A statistic multiplexed computing architecture generalizes the principles of packet switching at the **application level**.

• SMC has 3 basic requirements:
  – A well-defined unit of transmission(UT).
  – Store and Forward with re-transmission.
  – An application API with support of idempotent timeout re-transmission.
SMC Difference vs MPI:

• In MPI Every state transition is a single point of failure
• In Statistical Multiplexed Computing (SMC) we propose replicated computation state with multiple possible transition paths between states.
Architecture of Jenergy (SMC-based)

• We developed an HPC library based on the SMC ideas: Jenergy.

• Previous implementation had two drawbacks:
  – SPOF and bottleneck at the Tuple Space
    • Centralized and not replicated.
  – Used checkpoints for master fault tolerance
    • Experimental, Not portable, needed kernel support.

• We tackled these two elements:
  – Distributed replicated Tuple space
  – Checkpoint-less master fault tolerance
Architecture of Jenergy (SMC-based)
Architecture of Jenergy (SMC-based)

1. Distributed Data store
   - Cassandra, multi-master, eventually consistent, replicated, distributed storage.
   - Configured as first-write-wins to provide immutability of the data.

2. Distributed transactional queue
   - Zookeeper, distributed synchronization.
   - No single point of failure but not fully SMC
   - Performance reduction as the number of replicas increases and the write/read ratio increases.
   - Acceptable for our case since we only store the tuple ids
   - Since this involved data intensive architecture research, it is left unaddressed
Dealing with failures

• MPI stops entire job if it experiences any transient failure.

• In SMC transient failures do not stop the entire job.

• We examine the 2 most interesting types of failures scenarios:
  – Worker failure
  – Master failure
Worker Failure

Master
put A0,A1

Compute Space

W1
Get A
A0

W2
Get A
A1

Put C1

Get A
A0

Put C0

Init state

Staged state

Done state

Output tuple

Failure event
Master Failure

M1 Internal state

M1 Internal state gets lost after a failure

M1

put A0

ok

M2

put A0

Nack: Exists

put A1

ok

M2 Internal state is unaffected

M2 Internal state (eventually identical to M1)

M2 becomes primary master
**Supported Applications**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic</td>
<td>Supported.</td>
</tr>
<tr>
<td></td>
<td>e.g.: Linear algebra, spectral methods, n-body problems, finite element analysis,...</td>
</tr>
<tr>
<td>Non-Deterministic</td>
<td>Supported.</td>
</tr>
<tr>
<td></td>
<td>e.g.: Parallel search, unstable sorting, ...</td>
</tr>
<tr>
<td>Deterministic</td>
<td>Not Supported.</td>
</tr>
<tr>
<td></td>
<td>e.g.: Parallel search with random branching and data reduction.</td>
</tr>
<tr>
<td>Non-Deterministic</td>
<td>Supported.</td>
</tr>
<tr>
<td></td>
<td>e.g.: Monte-carlo simulations, date/time based applications.</td>
</tr>
</tbody>
</table>
Demo

• Worker failure:
  – http://www.youtube.com/watch?v=YziW_NLaBf8

• Master failure with dual masters:
  – http://www.youtube.com/watch?v=ZK8wcmh4Jkk

• Optionally:
  – Master failure with master restart
    • http://www.youtube.com/watch?v=Q4C_UUlrSKw
  – Dual masters with no failures
    • http://www.youtube.com/watch?v=3w0ylo7BZTA
  – Storage node failure
    • http://www.youtube.com/watch?v=haM4lw-EtOU
  – Queue node failure
    • http://www.youtube.com/watch?v=FXSgUaqY4Qc
Zero-Rollback Advantage with Component Failures

Time

Checkpoint

Transient Failures

Time(S)
Energy(MW)
Cost($)
Savings!

MPI/CPR

Start

Jenergy/SMC

Start

Statistical task multiplexing
Expected Performance Results

- **Expected time with no Failures**
- **MPI C/R = 5%**
- **Synergy C/R = 5%**

- **Expected time with no Failures**
- **MPI C/R = 25%**
- **Synergy C/R = 25%**

- **Expected time with no Failures**
- **MPI C/R = 50%**
- **Synergy C/R = 50%**

- **Expected time with no Failures**
- **MPI C/R = 75%**
- **Synergy C/R = 75%**
Actual Performance under Failures

![Graph showing actual performance under failures](image-url)
Performance with no failures

• Can we get close or better than failure-free MPI performance?

• Yes:
  – At larger data sizes
  – By tuning the tuples granularity
Scalability with Data Growth

Slowdown of Jenergy-TCP vs MPI-TCP (Matrix Multiply)

- Blue line: MPI-TCP-NoCPR
- Red line: Jenergy-TCP-C:Quorum

The graph shows the slowdown of the two communication protocols as the data size increases. The red line (Jenergy-TCP-C:Quorum) starts at a higher slow-down ratio but decreases rapidly as the data size increases, reaching a nearly constant value by 5000 data size. The blue line (MPI-TCP-NoCPR) remains relatively stable throughout the range of data sizes shown.
Granularity Tuning

Running Time Comparison with Granularity Tuning
(Jenergy-TCP vs MPI-TCP, Matrix Multiply, N=6000, 16 workers)
Limitations and Future Work

• More tests at larger scales (100x nodes)
• Needs a fully SMC distributed queue
• Experiments with lower computational intensity (md)
• Sharper definition of supported applications (for master with no CPR).
Conclusions

• Exascale computing make failures more frequent

• Auction-based clouds exacerbate the problem, but promise lower costs.

Envisioned Advantages of SMC:

  – Increasing processor counts without losing application reliability
  
  – Improving HPC application’s energy efficiency by eliminating energy waste in checkpoints and rolled back calculations
Acknowledgments

CNS0958854

NSF

TEMPLE UNIVERSITY®

amazon web services™

of the City of Philadelphia

TeraGrid™

ICMS

Institute for Computational Molecular Science

XSEDE

Extreme Science and Engineering Discovery Environment
Backup slides for AHPC

Exploration of Auction-based HPC clouds:

– Developed a method for bid-aware checkpoint intervals
– Discussed the effect of bid-price coupling on the failure model
– Developed a model to determine the runtime/cost of parallel applications under various bidding strategies
Auction-based HPC Clouds

• More **Resource Volatility** than regular clusters
• Market-driven Environment
• Conceptual study
Auction-Based Cloud Computing

- The fairest pricing model (buyer’s perspective)
- The best way to optimize resource efficiency (seller’s perspective).
Main Research Questions

1. How to determine market-aware checkpoint intervals?
   – We developed a model for *bid-aware checkpoint intervals*.

2. What is the effect of bidding strategies on the failure model?
   – Exposed the problem of *bid price coupling and correlated failures*
   – Proposed a *multi-tier bidding method*.

3. How to determine the runtime/cost of parallel applications under various bidding strategies?
   – Leveraged the steady state timing model and produced a *market-aware probabilistic timing model*
Bidding strategies

• In AHPC Failures rates are based on two things:
  – Market prices
  – Bid of the user

• How to determine bid-aware checkpoint intervals?
Bidding strategies

• How to extract the failure rate from the market history?

Fig. 2: CDF of price probability per bid price for cc2.8xlarge instances
Bidding strategies

• We developed a bid-aware checkpoint interval model for **coupled** applications.

• Expected Runtime:

\[ E_T = \frac{T}{t_0}(K_0 + t_0 + \alpha_{bid_i}(t_0(K_1 + K_2) + \frac{t_0^2}{2})) \]

• Corresponding Checkpoint Interval

\[ t_0 = \sqrt{\frac{2K_0}{\alpha_{bid_i}}} \]
Theoretical Evaluation

The optimal checkpoint interval provides good protection in case of out-of-bid failures compared to other interval choices.
Bidding strategies

• How about **decoupled** programming models?
• We noticed that there is an **upstream** problem
• The bidding strategy affects the failure rate as well as the **failure model**.
Spot instances and Failure models

- Traditional Cluster Failure model
- At most 10% of nodes down at one time.
Spot instances and Failure models

- Given **Uniform bid** on all cluster instances:
- An out-of-bid failure can take down whole cluster at once!

**Bid-Price Coupling. Bid price becomes SPOF!**
Bidding strategies

• Uniform bidding can trigger a total failure.
• Even if decoupled models can deal with partial failures, **bid-price coupling** invalidates this good property.
• We proposed a model for **multi-tier bidding strategies**
Spot instances and Failure models

• We explored the idea of non-uniform bidding
• Each instance can have its own bid price
  – Leading to “separate” out-of-bid failure rates
Probabilistic Timing Model (PTM)

Includes the market information and the user’s bid
PTM Usage

- Price History
- Generate bids Strategies
- Estimate ET, EP
- Statistical Modeling
- Obtain CDF Function
Example of Bid selection (Deadline)

Expected Elapsed Time vs. 10 Bidding strategies with 3 Classes

Only bid strategies above (mid, mid, mid) can meet the deadline of scenario

Scenario deadline (minimum time)

Fig. 9: Expected elapsed time vs 10 bidding strategies with 3 classes
Example of Bid selection (Cost-deadline)

This is the BUDGET for our scenario.

Only these bid strategies can be used.

This is the lowest expected time.

Advisable Bidding strategy: Saving of 48% compared to BUDGET.

Fig. 10: Expected total maximum cost vs 10 bidding strategies with 3 classes.