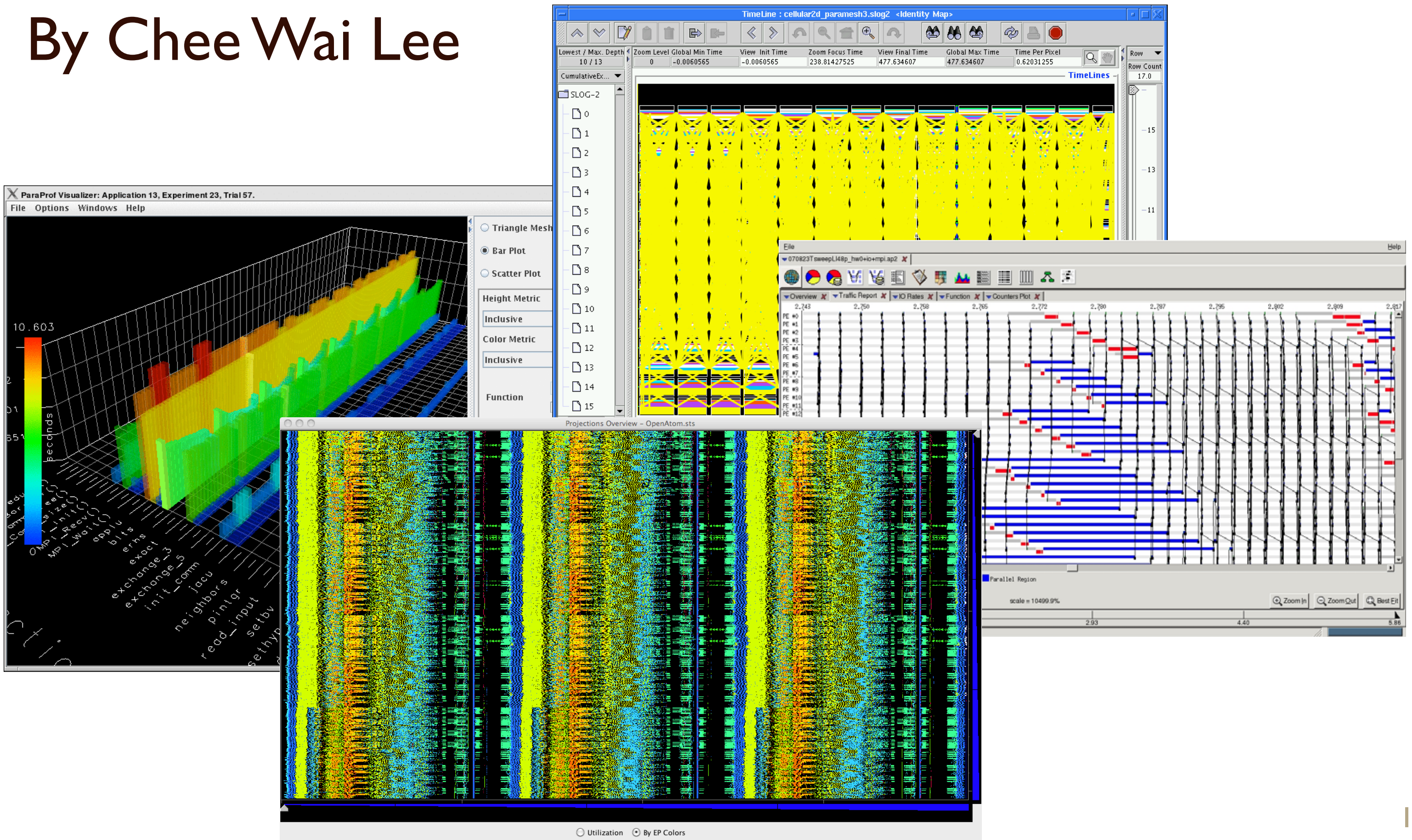
Techniques in Scalable and Effective Performance Analysis

Thesis Defense - 11/10/2009

By Chee Wai Lee



Overview

- Introduction.
- Scalable Techniques:
  - Support for Analysis Idioms
  - Data Reduction
  - Live Streaming
  - Hypothesis Testing
- Conclusion.

Introduction

- What does performance analysis of applications with visual tools entail?
- What are the effects of application scaling on performance analysis?

Effects of Application Scaling

- Enlarged performance-space.
- Increased performance data volume.
- Reduces accessibility to machines and increases resource costs
  - Time to queue.
  - CPU resource consumption.

Main Thrusts

- Tool feature support for Scalable Analysis Idioms.
- Online reduction of performance data volume.
- Analysis Idioms for applications through live performance streaming.
- Effective repeated performance hypothesis testing through simulation.

Main Thrusts

- Tool feature support for Scalable Analysis Idioms.
- Online reduction of performance data volume.
- Analysis Idioms for applications through live performance streaming.
- Effective repeated performance hypothesis testing through simulation.

Scalable Tool Features: Motivations

- Performance analysis idioms need to be effectively supported by tool features.
- Idioms must avoid using tool features that become ineffectual at large processor counts.
- We want to catalog common idioms and match these with scalable features.

Scalable Tool Feature Support (1/2)

- Non-scalable tool features require analysts to **scan** for visual cues over the processor domain.

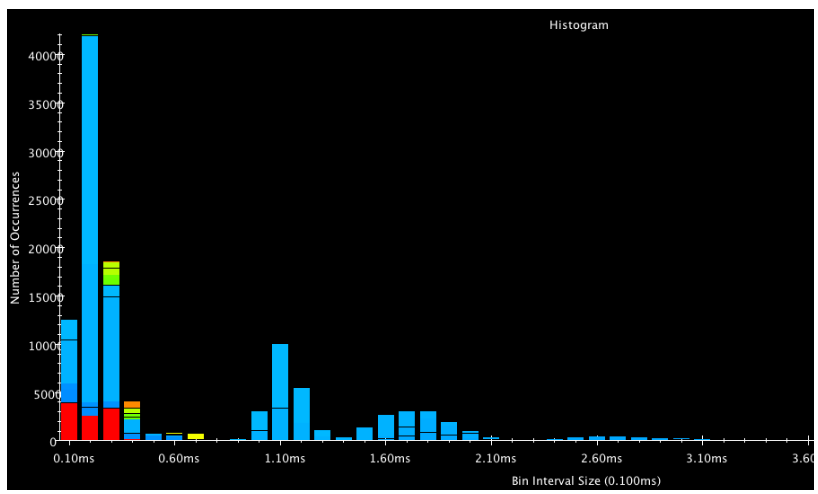
- How do we avoid this requirement on analysts?

Scalable Tool Feature Support (2/2)

- Aggregation across processor domain:
  - Histograms.
  - High resolution Time Profiles.
- Processor selection:
  - Extrema Tool.

Histogram as a Scalable Tool Feature

- Bins represent time spent by activities.
- Counts of activities across all processors are added to appropriate bins.
- Total counts for each activity are displayed as different colored bars.

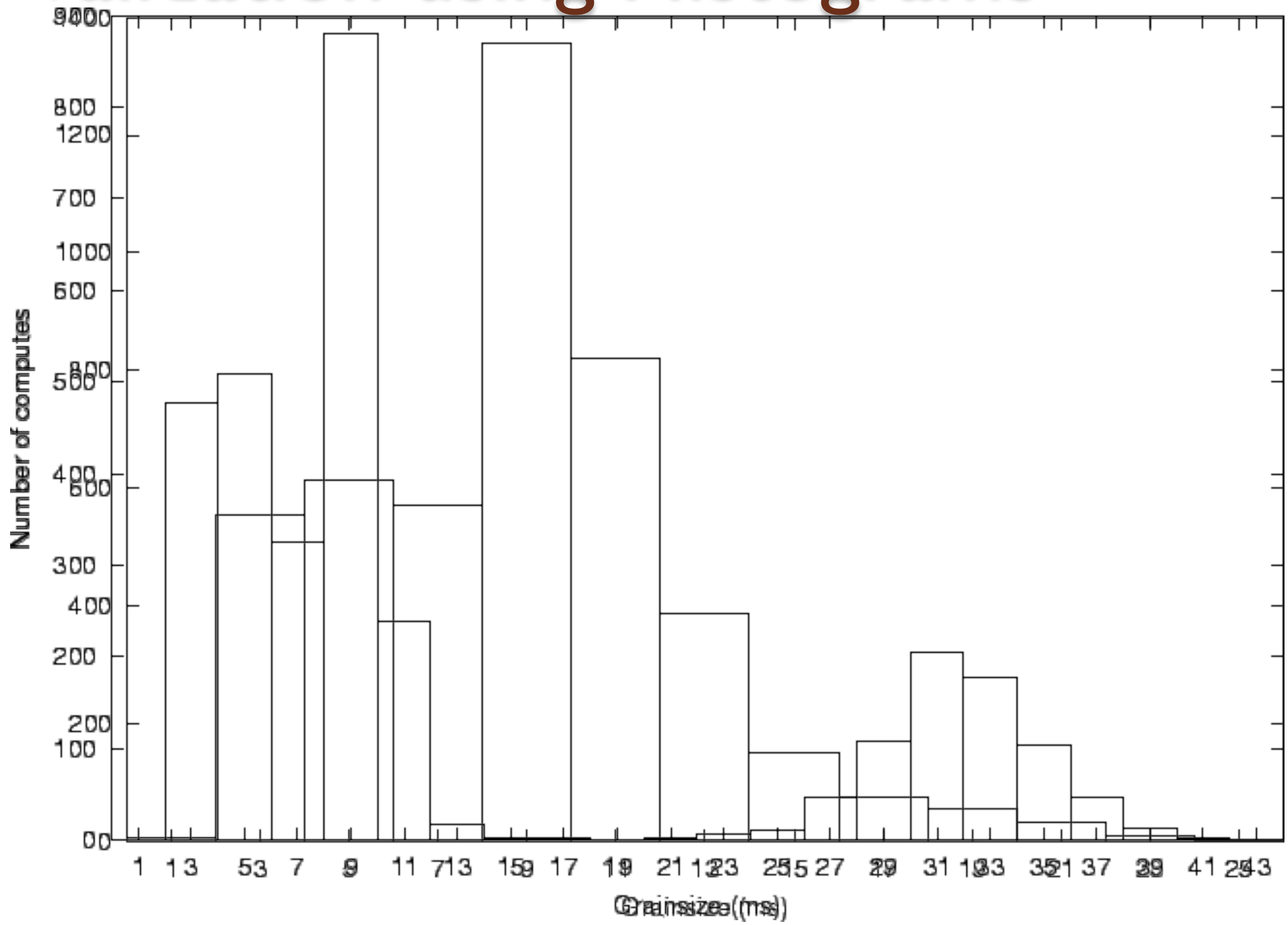


Case Study:

- Apparent load imbalance.
- No strategy appeared to solve imbalance.
- Picked overloaded processor timelines.*
- Found longer-than-expected activities.
- Longer activities associated with specific objects.
- Possible work grainsize distribution problems.

*As we will see later, not effective with large numbers of processors.

Case Study: Validation using Histograms



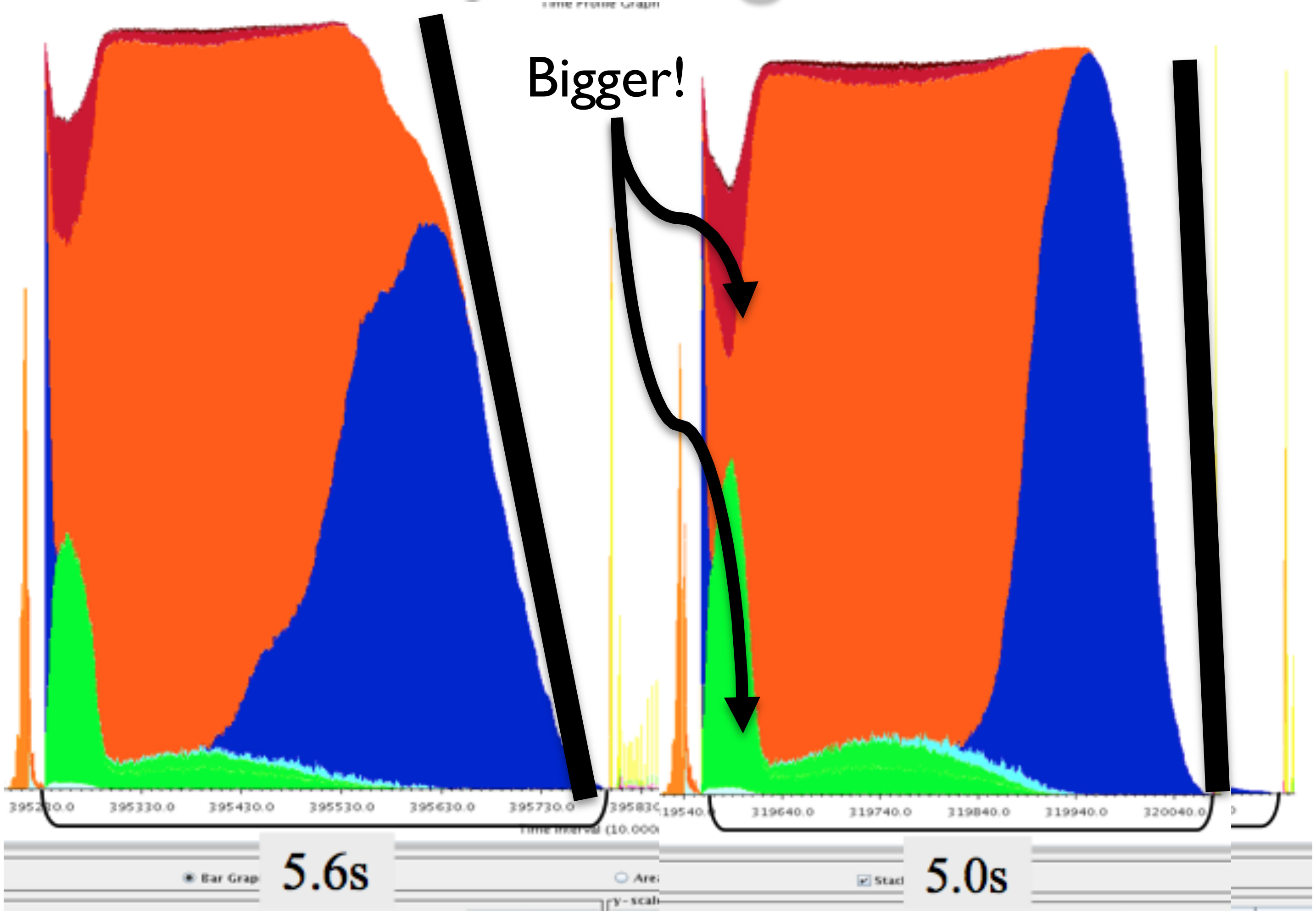
Effectiveness of Idiom

- Need to find way to pick out overloaded processors. Not scalable!
- Finding out if work grainsize was a problem simply required the histogram feature.

High Resolution Time Profiles

- Shows activity-overlap over time summed across all processors.
- Heuristics guide the search for visual cues for various potential problems:
  - Gradual downward slopes hint at possible load imbalance.
  - Gradual upward slopes hint at communication inefficiencies.
- At high resolution, gives insight into application sub-structure.

Case Study: Using Time Profiles



Possible Load Imbalance

After Greedy Load Balancing Strategy

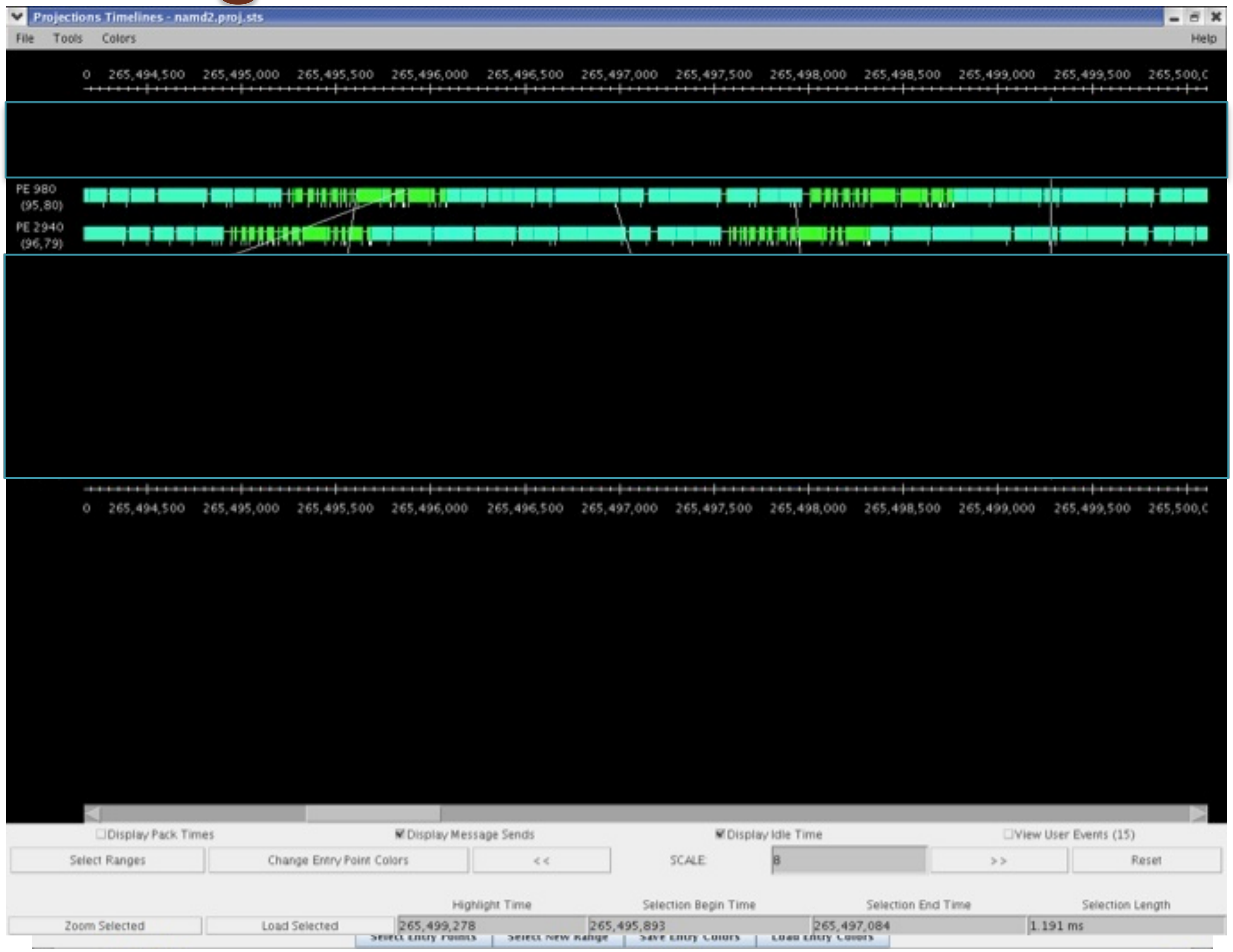
Finding Extreme or Unusual Processors

- A recurring theme in analysis idioms.
- Easy to pick out timelines in datasets with small numbers of processors.
- Examples of attributes and criteria:
  - Least idle processors.
  - Processors with late events.
  - Processors that behave very differently from the rest.

The Extrema Tool

- Semi-automatically picks out interesting processors to display.
- Decisions based on analyst-specified criteria.
- Mouse-clicks on bars load interesting processors onto timeline.

Using the Extrema Tool



Scalable Tool Features: Conclusions

- Effective analysis idioms must avoid non-scalable features.
- Histograms, Time Profiles and the Extrema Tool offer scalable features in support of idioms.

Main Thrusts

- Tool feature support for Scalable Analysis Idioms.
- Online reduction of performance data volume.
- Analysis Idioms for applications through live performance streaming.
- Effective repeated performance hypothesis testing through simulation.

Data Reduction

- Normally, scalable tool features are used with full event traces.
- What happens if full event traces get too large?
- We can:
  - Choose to keep event traces for only a subset of processors.
  - Replace event traces of discarded processors with interval-based profiles.

Interval-Based Profiles

- Small files. File size is a function of duration of instrumentation and resolution of each time interval recorded.

- Suitable for Time Profiles.

Choosing Useful Processor Subset (1/2)

- What are the challenges?
  - No a priori information about performance problems in dataset.
  - Chosen processors need to capture details of performance problems.

Choosing Useful Processor Subsets (2/2)

- Observations:
  - Processors tend to form equivalence classes with respect to performance behavior.
  - Clustering can be used to discover equivalence classes in performance data.
  - Outliers in clusters may be good candidates for capturing performance problems.

Applying *k*-Means Clustering to Performance Data (1/2)

- *k*-Means Clustering algorithm is commonly used to classify objects in data mining applications.
- Treat the vector of recorded performance metric values on each processor as a data point for clustering.

Applying *k*-Means Clustering to Performance Data (2/2)

- Measure similarity between two data points using the Euclidean Distance between the two metric vectors.

- Given *k* clusters to be found, the goal is to minimize similarity values between all data points and the centroids of the *k* clusters.

Choosing from Clusters

- Choosing Cluster Outliers.
  - Pick processors furthest from cluster centroid.
  - Number chosen by proportion of cluster size.
- Choosing Cluster Exemplars.
  - Pick a single processor closest to the cluster centroid.
- Outliers + Exemplars = Reduced Dataset.

Applying *k*-Means Clustering Online

- Decisions on data retention are made before data is written to disk.

- Requires a low-overhead and scalable parallel *k*-Means algorithm which was implemented.

Parallel *k*-Means

Root	**Worker**
	Contribute metric vector.
Receive aggregated metric vector stats. Calculate normalization factors. Get initial cluster centroids. **Broadcast** factors and centroids.	
	Normalize local metric vector. Find closest centroid. **Contribute** centroid modification.

Update centroids.
If no centroid changes,
   Done
Else
   Broadcast centroids	

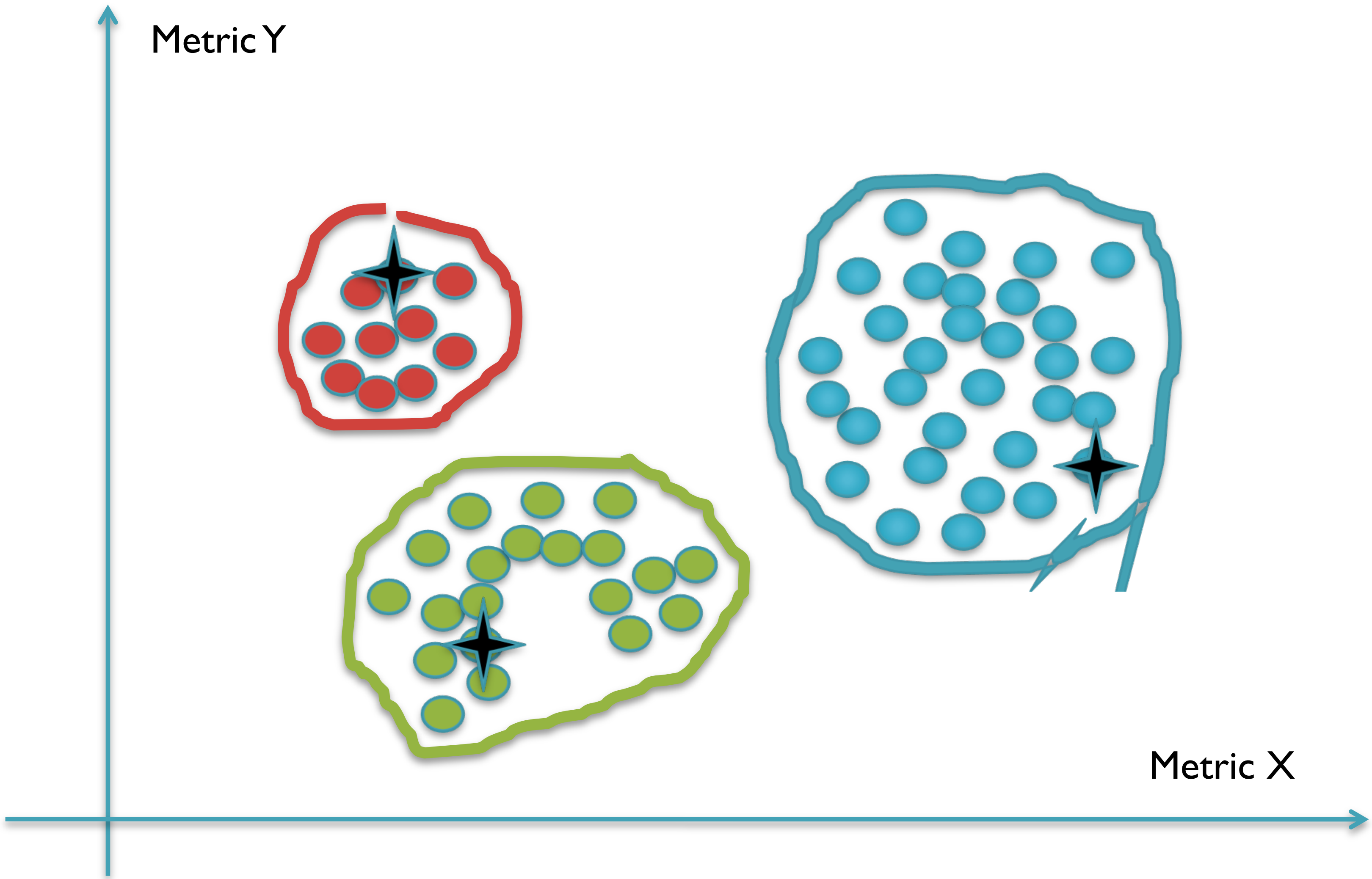
Important *k*-Means Parameters

- Choice of metrics from domains:
  - Activity time.
  - Communication volume (bytes).
  - Communication (number of messages).
- Normalization of metrics:
  - Same metric domain = no normalization.
  - *Min-max* normalization across different metric domains to remove inter-domain bias.

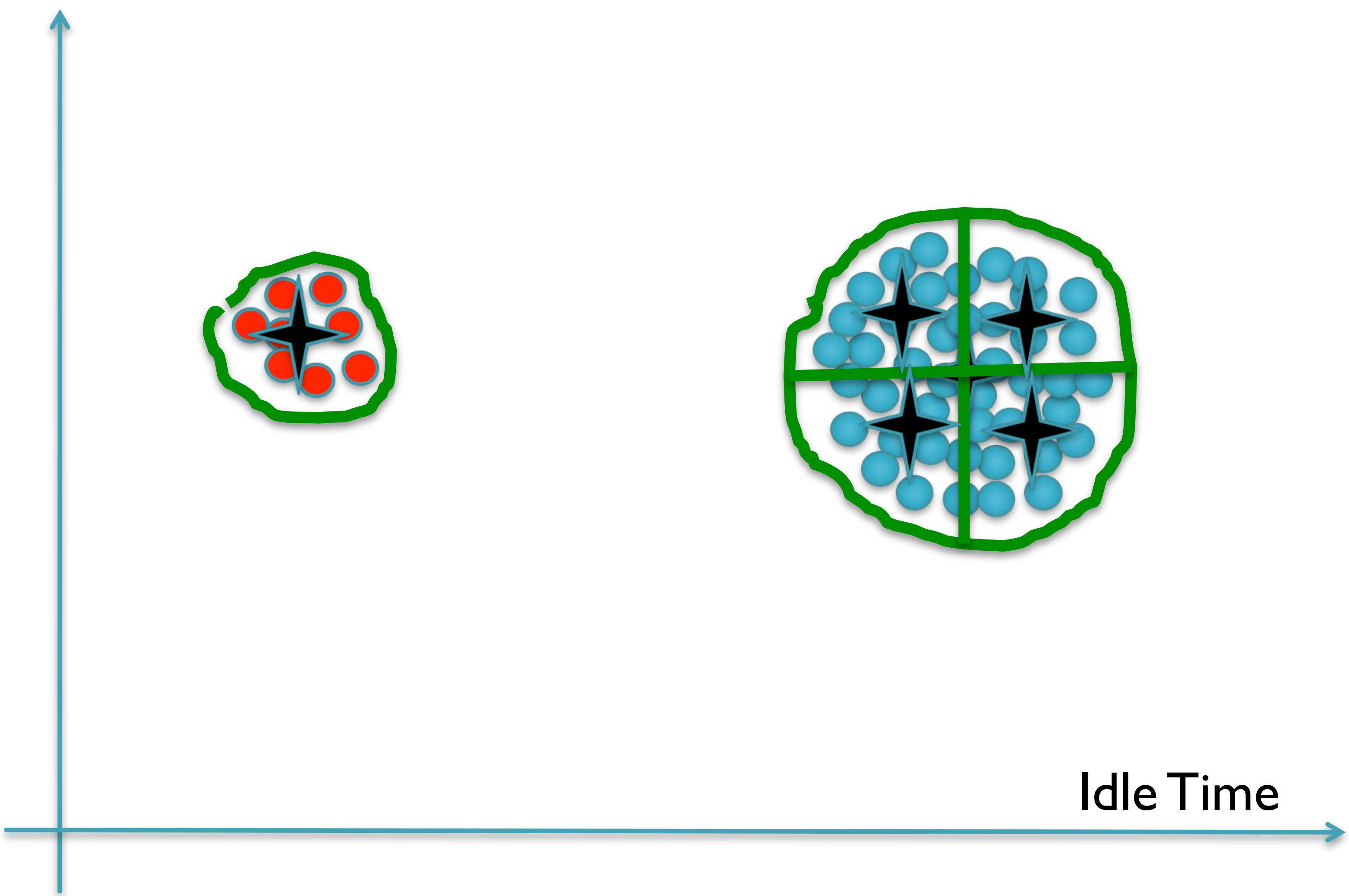
Min-Max Normalization for Multiple Metric Domains

- Find min_m values for each metric m over all processor data points.
- Find max_d values for metrics within each metric domain d over all processor data points.
- For each data point, re-compute each metric value m, where m is a member of domain d, as: $(m - min_m)/max_d$

k-Means Clustering



Clustering Nuances



Evaluating the technique

- Clustering and choice heuristics presented us with a reduced dataset.
- How useful is the reduced dataset to analysis?
- We know least-idle processors can be useful for analysis.
- How many top least-idle processors will show up in the reduced dataset?
- What was the overhead?

Results (2048 Processors NAMD)

Percentage of Top Least Idle processors picked for the reduced dataset.

Top x Least Idle	5% Retention	10% Retention	15% Retention
5	100%	100%	100%
10	70%	90%	100%
20	45%	70%	95%

5% Retention = 102 processors
10% Retention = 204 processors
15% Retention = 306 processors

Results (1024 Processors NAMD)

Percentage of Top Least Idle processors picked for the reduced dataset.

Top x Least Idle	5% Retention	10% Retention	15% Retention
5	20%	40%	60%
10	20%	40%	50%
20	10%	20%	30%

5% Retention = 51 processors
10% Retention = 102 processors
15% Retention = 153 processors

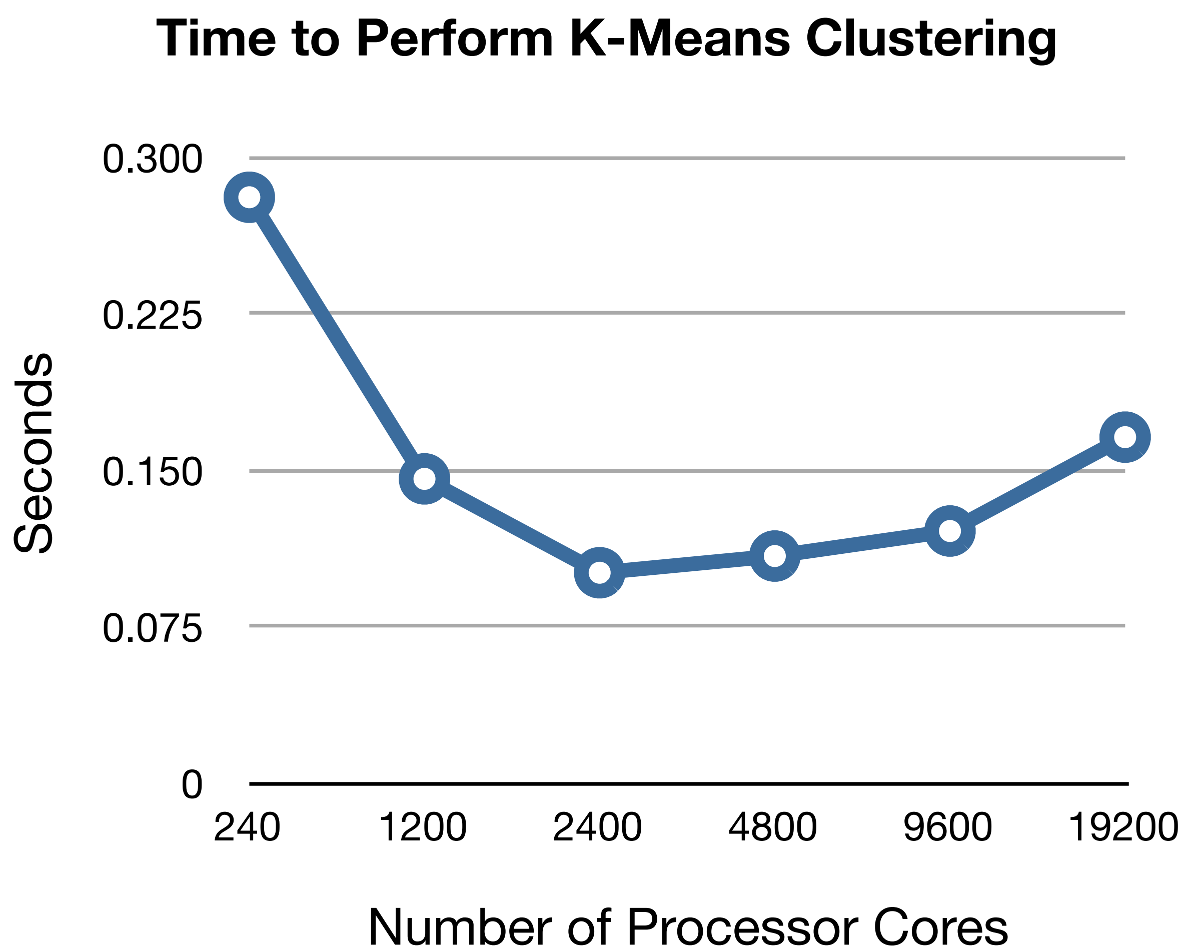
Results (4096 Processors NAMD)

Percentage of Top Least Idle processors picked for the reduced dataset.

Top x Least Idle	2.5% Retention	5% Retention	7.5% Retention
5	40%	100%	100%
10	20%	70%	100%
20	10%	45%	100%

2.5% Retention = 102 processors
5% Retention = 204 processors
7.5% Retention = 306 processors

Overhead of parallel k-Means



Data Reduction: Conclusions

- Showed combination of techniques for online data reduction is effective*.
- Choice of processors included in reduced datasets can be refined and improved
  - Include communicating processors.
  - Include processors on critical path.
- Consideration of application phases can further improve quality of reduced dataset.

*Chee Wai Lee, Celso Mendes and Laxmikant V. Kale. **Towards Scalable Performance Analysis and Visualization through Data Reduction.** 13th International Workshop on High-Level Parallel Programming Models and Supportive Environments, Miami, Florida, USA, April 2008.

Main Thrusts

- Tool feature support for Scalable Analysis Idioms.
- Online reduction of performance data volume.
- Analysis Idioms for applications through live performance streaming.
- Effective repeated performance hypothesis testing through simulation.

Live Streaming of Performance Data

- Live Streaming mitigates need to store a large volume of performance data.
- Live Streaming enables analysis idioms that provide animated insight into the trends application behavior.
- Live Streaming also enables idioms for the observation of unanticipated problems, possibly over a long run.

Challenges to Live Streaming

- Must maintain low overhead for performance data to be recorded, pre-processed and disposed-of.
- Need efficient mechanism for performance data to be sent via out-of-band channels to one (or a few) processors for delivery to a remote client.

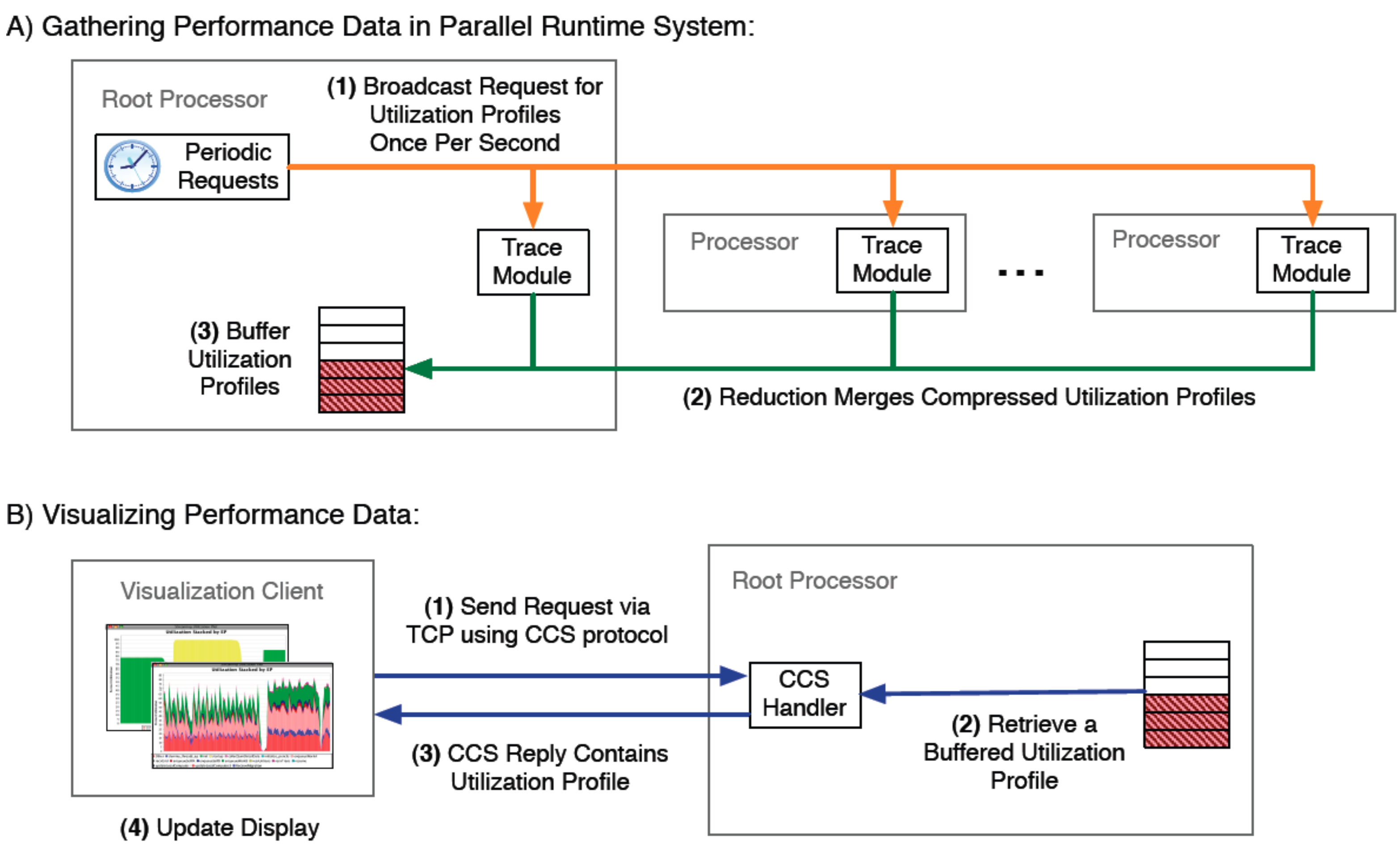
Enabling Mechanisms

- Charm++ adaptive runtime as medium for scalable and efficient:
  - Control signal delivery.
  - Performance data capture and delivery.
- Converse Client-Server (CCS) enables remote interaction with running Charm++ application through a socket opened by the runtime.

Questions

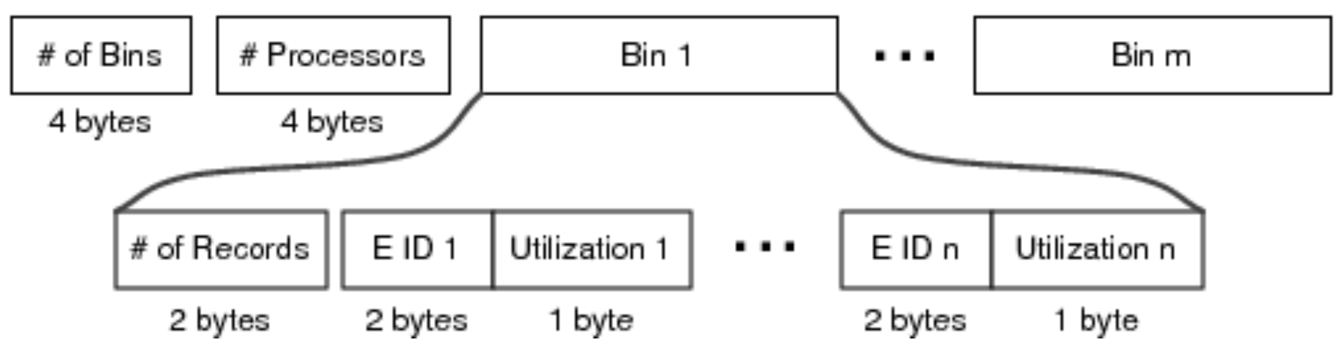
- What kinds of performance data should we stream?
- How frequently should we deliver the data to the client?

Live Streaming System Overview

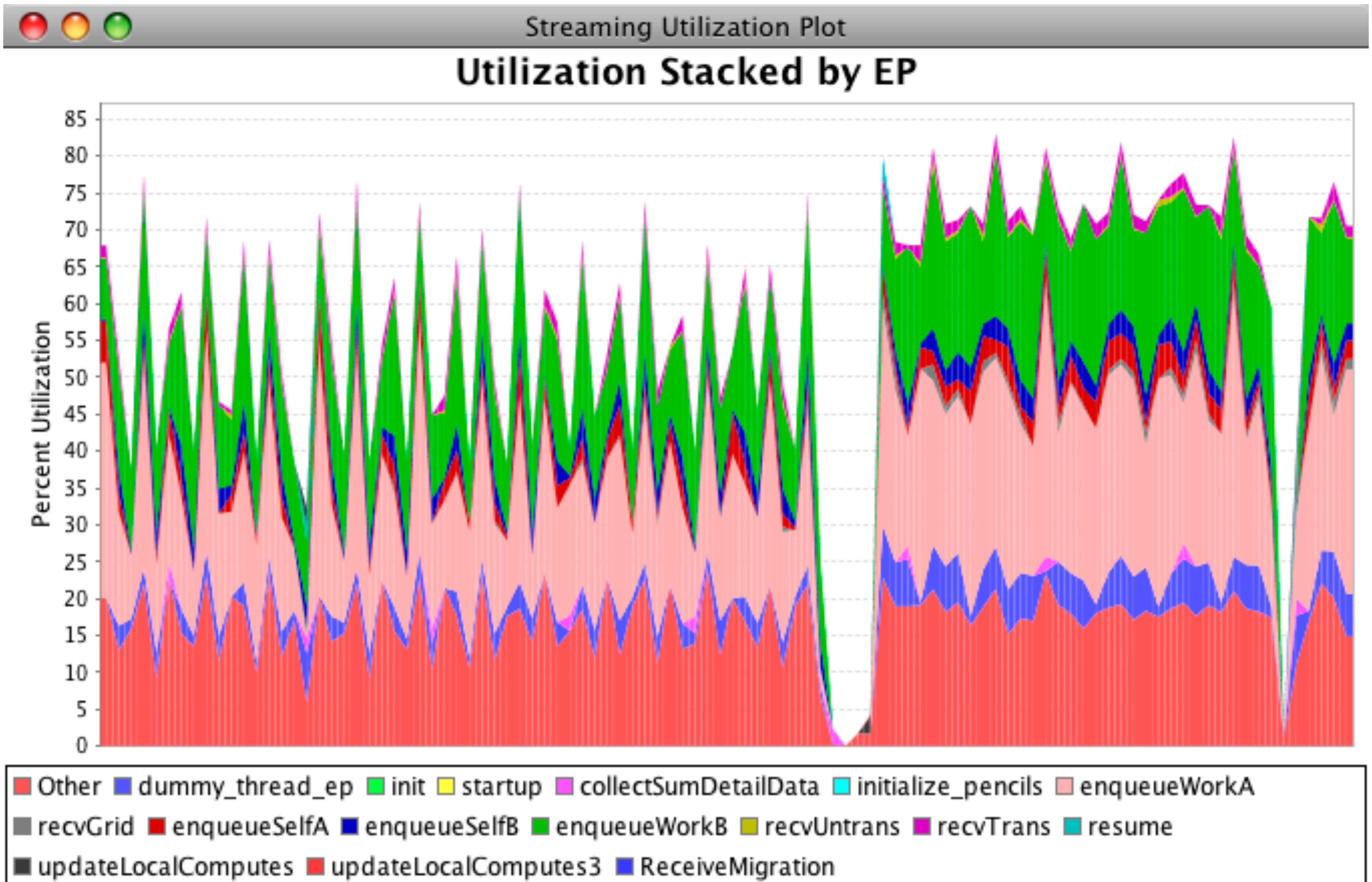


What is Streamed?

- A Utilization Profile similar to high resolution Time Profiles.
- Performance data is compressed by only considering significant metrics in a special format.
- Special reduction client merges data from multiple processors.



Visualization



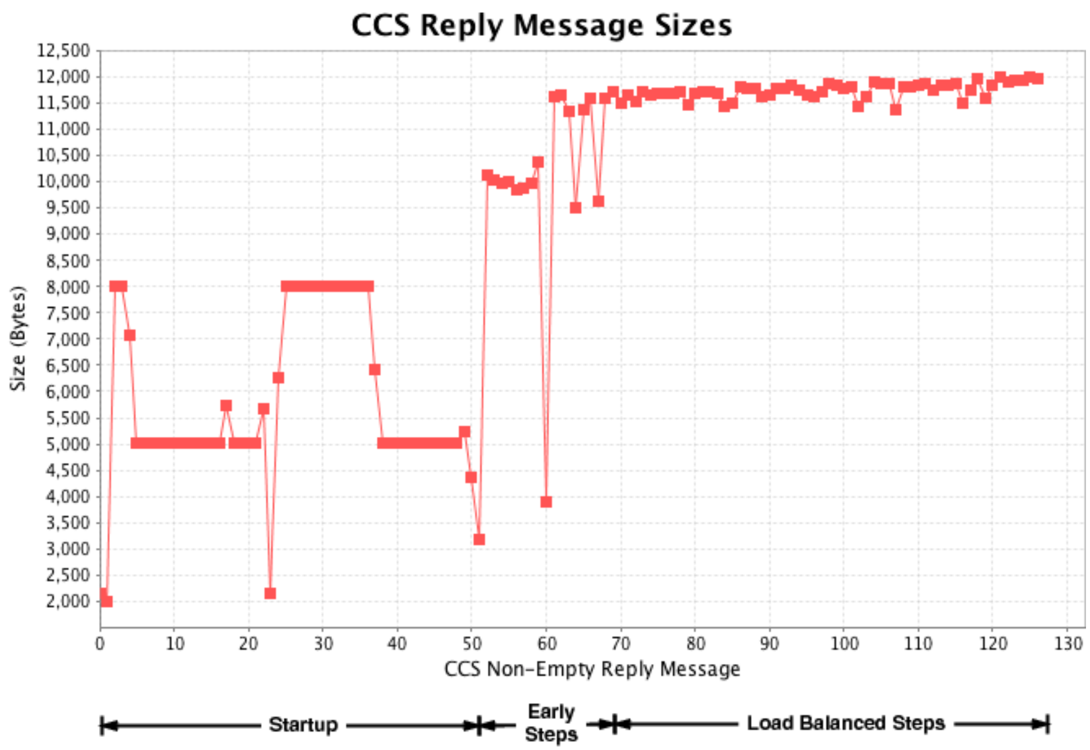
Overheads (1/2)

% Overhead when compared to baseline system: Same application with no performance instrumentation.

	512	1024	2048	4096	8192
With instrumentation, data reductions to root **with remote client** attached.	0.94%	0.17%	-0.26%	0.16%	0.83%
With instrumentation, data reductions to root but **no remote client** attached.	0.58%	-0.17%	0.37%	1.14%	0.99%

Overheads (2/2)

For bandwidth consumed when streaming performance data to the remote visualization client.



Live Streaming: Conclusions*

- Adaptive runtime allowed out-of-band collection of performance data while in user-space.
- Achieved with very low overhead and bandwidth requirements.

*Isaac Dooley, Chee Wai Lee, and Laxmikant V. Kale. **Continuous Performance Monitoring for Large-Scale Parallel Applications**. Accepted for publication at HiPC 2009, December-2009.

Main Thrusts

- Tool feature support for Scalable Analysis Idioms.
- Online reduction of performance data volume.
- Analysis Idioms for long-running applications through live performance streaming.
- Effective repeated performance hypothesis testing through simulation.

Repeated Large-Scale Hypothesis Testing

- Large-Scale runs are expensive:
  - Job submission of very wide jobs to supercomputing facilities.
  - CPU resources consumed by very wide jobs.

- How do we make repeated but inexpensive hypothesis testing experiments?

Trace-based Simulation

- Capture event dependency logs from a baseline application run.

- Simulation produces performance event traces from event dependency logs.

Advantages

- The time and memory requirements at simulation time are divorced from requirements at execution time.
- Simulation can be executed on fewer processors.
- Simulation can be executed on a cluster of workstations and still produce the same predictions.

Using the BigSim Framework (1/2)

- BigSim emulator captures:
  - Relative event time stamps.
  - Message dependencies.
  - Event dependencies.
- BigSim emulator produces event dependency logs.

Using the BigSim Framework (2/2)

- BigSim simulator uses a PDES engine to process event dependency logs to predict performance.
- BigSim simulator can generate performance event traces based on the predicted run.

Examples of Hypothesis Testing Possible

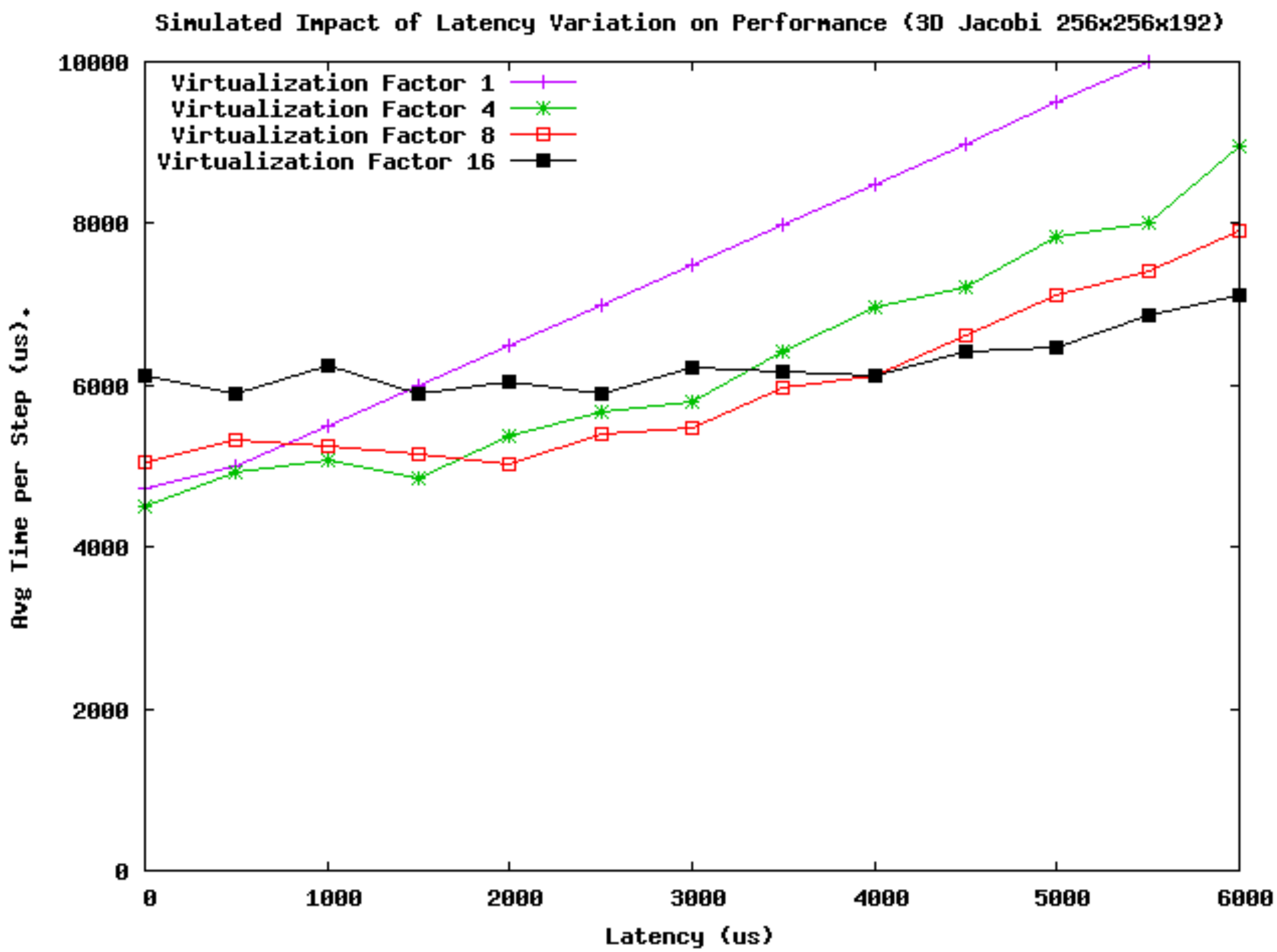
- Hypothetical Hardware changes:
  - Communication Latency.
  - Network properties.

- Hypothetical Software changes:
  - Different load balancing strategies.
  - Different initial object placement.
  - Different number of processors with the same object decomposition.

Example: Discovering Latency Trends

- Study the effects of network latency on performance of seven-point stencil computation.

Latency Trends – Jacobi 3d 256x256x192 on 48 pes



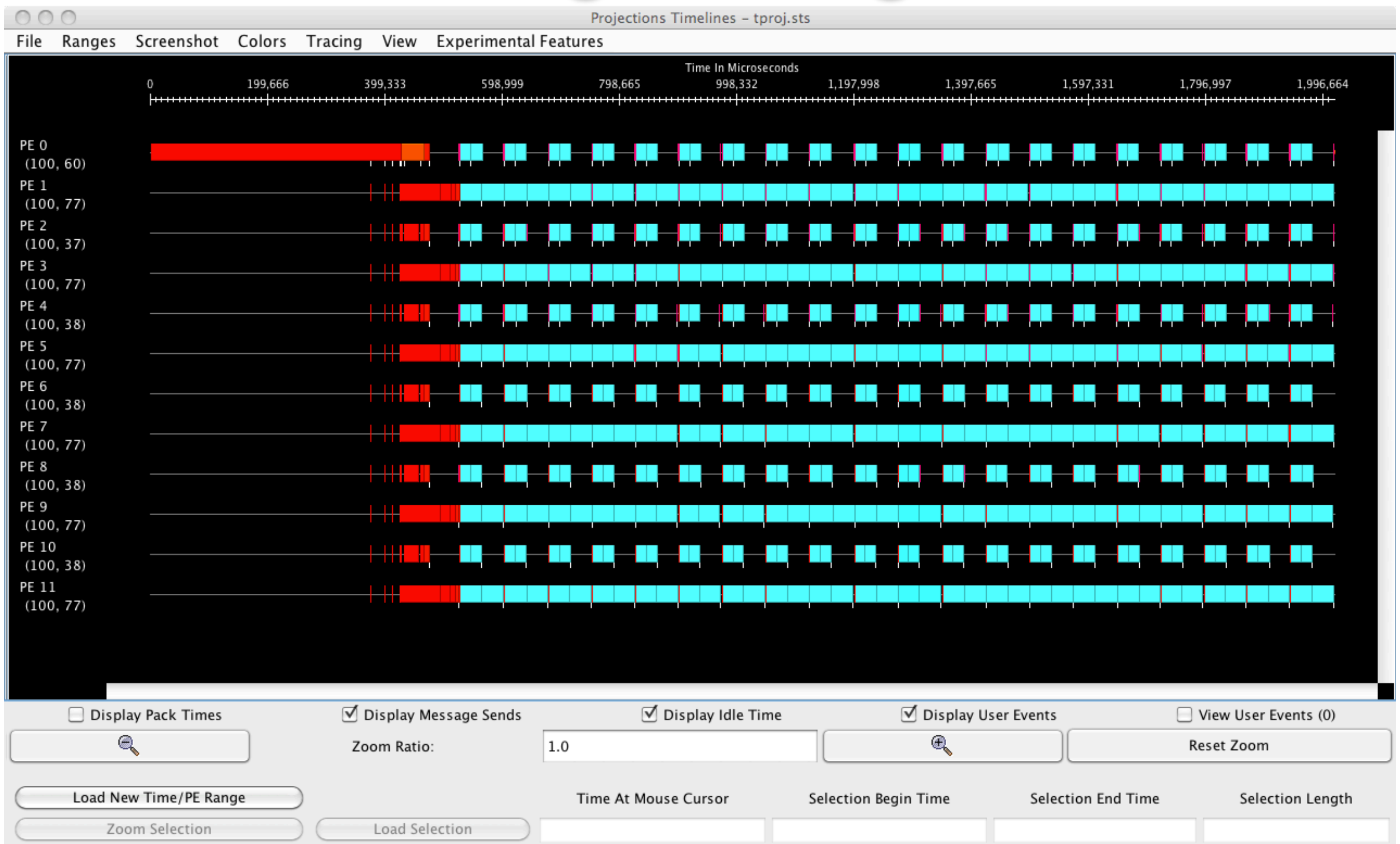
Testing Different Load Balancing Strategies (1/2)

- Load Balancing Strategies make decisions as object-to-processor maps based on object load and inter-object communication costs.
- How do we make the simulator produce predictions about new load balancing strategies without re-executing the original code?

Testing Different Load Balancing Strategies (2/2)

- Record object-load and communication information of baseline run.
- Different Load Balancing strategies create different object-to-processor maps.
- A log transformation tool I wrote, transforms event dependency logs to reflect new object-to-processor mapping.

Example:
Load Balancing Strategies



Baseline Strategy: Half the processors perform twice the work with 2 objects per processor.

Greedy Strategy: Objects balanced across processors perfectly.

Reduction of Processors during Emulation

- BigSim emulator can emulate ***k*** processors on ***p*** physical processors
- Ratio of ***k*** to ***p*** can be increased by memory aliasing where appropriate.

Hypothesis Testing: Conclusions

- Flexible repeated performance hypothesis testing can be achieved via trace-based simulation.
- No analytical models need to be constructed for each application to enable software changes such as load balancing strategies.

Extending Scalability Techniques

- Can the techniques described in this thesis be adopted by other tools quickly?
- This was investigated through the results of a collaboration with the TAU group*.
- Flexible Performance call-back interface in Charm++ enabled an easy mechanism for a popular tool like TAU to record and process key runtime and application events.

*Scott Biersdorff, Chee Wai Lee, Allen D. Malony and Laximkant V. Kale**.** **Integrated Performance Views in Charm++: Projections Meets TAU**. ICPP-2009, Vienna, Austria, September 22-25, 2009.

Benefits of Extension of Capabilities

- Scalable TAU tools features can be used to grant different performance insights into Charm++ applications.
- TAU can make use of the adaptive runtime for live streaming of TAU data.
- TAU can make use of BigSim for repeated hypothesis testing.

Thesis Contributions (1/2)

- Identified and developed scalable tool feature support for performance analysis idioms.
- Showed the combination of techniques and heuristics effective for data reduction.
- Showed how an adaptive runtime can efficiently stream live performance data out-of-band in user-space to enable powerful analysis idioms.

Thesis Contributions (2/2)

- Showed trace-based simulation to be an effective method for repeated hardware and software hypothesis testing.
- Highlighted importance of flexible performance frameworks for the extension of scalability features to other tools.