MetaBalancer: An automatic load balancer based on application characteristics

Harshitha Menon

UIUC

7^{th} May, 2012

Outline

1 Motivation

2 Meta-Balancer: Overview

3 Load Balancer: Existing Framework

4 Meta-Balancer
- Statistics Collection
- Ideal LB Period
- Strategy Selection

5 Conclusion

6 Future Work

Outline

1 Motivation

2 Meta-Balancer: Overview

3 Load Balancer: Existing Framework

4 Meta-Balancer
- Statistics Collection
- Ideal LB Period
- Strategy Selection

5 Conclusion

6 Future Work

Motivation

- Load balancing decisions depend on application
  - Multiple runs required to observe and decide
  - Tough to judge the correct load balancing parameters

Motivation

- Load balancing decisions depend on application
  - Multiple runs required to observe and decide
  - Tough to judge the correct load balancing parameters
- Dynamic applications require dynamic load balancing decisions
  - Some phases may need frequent load balancing, others may be static
  - Computation to communication ratio may change

Outline

1 Motivation

2 Meta-Balancer: Overview

3 Load Balancer: Existing Framework

4 Meta-Balancer
- Statistics Collection
- Ideal LB Period
- Strategy Selection

5 Conclusion

6 Future Work

Meta-Balancer

- Charm++ RTS monitors applications
  - Computation and communication per chare is maintained
  - RTS maintains and controls the placement of chares

Meta-Balancer

- Charm++ RTS monitors applications
  - Computation and communication per chare is maintained
  - RTS maintains and controls the placement of chares
- Charm++ RTS is aware of the system characteristics

Meta-Balancer

- Charm++ RTS monitors applications
  - Computation and communication per chare is maintained
  - RTS maintains and controls the placement of chares
- Charm++ RTS is aware of the system characteristics
- Offload the load balancing related decision making to Charm++ RTS
- Meta-Balancer makes load balancing decisions without any user involvement

Decisions in Meta-Balancer

- Frequency of load balancing

Decisions in Meta-Balancer

- Frequency of load balancing
- Adaptive triggering of load balancing

Decisions in Meta-Balancer

- Frequency of load balancing
- Adaptive triggering of load balancing
- Strategy Selection
  - Communication vs Computation strategy
  - Comprehensive vs Refinement strategy

Outline

1 Motivation

2 Meta-Balancer: Overview

3 Load Balancer: Existing Framework

4 Meta-Balancer
- Statistics Collection
- Ideal LB Period
- Strategy Selection

5 Conclusion

6 Future Work

Existing Framework

- User decides LB frequency and strategy

Existing Framework

- User decides LB frequency and strategy
- Control flow
  1. AtSync called whenever load balancing is to be performed in the application
  2. RTS enforces a chare level local barrier within every processor
  3. Global barrier to collect statistics

Existing Framework

- User decides LB frequency and strategy
- Control flow
  1. AtSync called whenever load balancing is to be performed in the application
  2. RTS enforces a chare level local barrier within every processor
  3. Global barrier to collect statistics
  4. Execute load balancing strategy and perform migration
  5. Application resumes

Outline

1 Motivation

2 Meta-Balancer: Overview

3 Load Balancer: Existing Framework

4 Meta-Balancer
- Statistics Collection
- Ideal LB Period
- Strategy Selection

5 Conclusion

6 Future Work

Lifecycle

- Periodically during an application run

Lifecycle

- Periodically during an application run
  1. Every processor contributes its statistics

Lifecycle

- Periodically during an application run
  1. Every processor contributes its statistics
  2. Based on the statistics collected, the central processor (root)
     - Finds the ideal LB period and informs other processors
     - If immediate LB required, informs other processors

Lifecycle

- Periodically during an application run
  1. Every processor contributes its statistics
  2. Based on the statistics collected, the central processor (root)
     - Finds the ideal LB period and informs other processors
     - If immediate LB required, informs other processors
  3. During load balancing, root decides the LB strategy

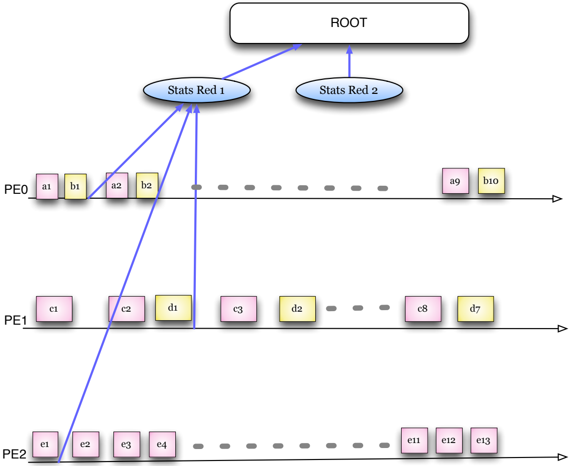
Asynchronous Collection of Stats via Reduction

- Statistics are collected via reduction periodically and frequently
- Collection has to be asynchronous - presence of a frequent local and global barrier results in substantial overheads

Asynchronous Collection of Stats via Reduction

- Statistics are collected via reduction periodically and frequently
- Collection has to be asynchronous - presence of a frequent local and global barrier results in substantial overheads
- Only minimal statistics are collected via custom reduction in Charm++
  - Maximum load - *max* reducer over all processor's load
  - Average load - *sum* reducer over all processor's load
  - Minimum Utilization - *min* reducer over all processor's utilization (ratio of busy time and total time)

Asynchronous Collection of Stats via Reduction



Ideal LB Period

- Load balancing removes load imbalance, but causes following overheads:
  - Data collection and strategy cost
  - Migration cost

Ideal LB Period

- Load balancing removes load imbalance, but causes following overheads:
  - Data collection and strategy cost
  - Migration cost
- Optimal performance obtained if load balancing is performed at an ideal period
- Gains obtained from load balancing is maximized despite the incurred overheads.

Ideal LB Period

Assuming,

τ - ideal LB period, γ - total iterations

Γ - execution time, θ - cost of LB

$y = ax + c_a$ - average load line equation

$y = mx + c_m$ - maximum load w.r.t average load

Ideal LB Period

Assuming,

τ - ideal LB period, γ - total iterations

Γ - execution time, θ - cost of LB

$y = ax + c_a$ - average load line equation

$y = mx + c_m$ - maximum load w.r.t average load

We obtain total execution time as

$\Gamma = \frac{\gamma}{\tau} \times (\int_0^{\tau} (mx + c_m)dx + \theta) + \int_0^{\gamma} (ax + c_a)dx$

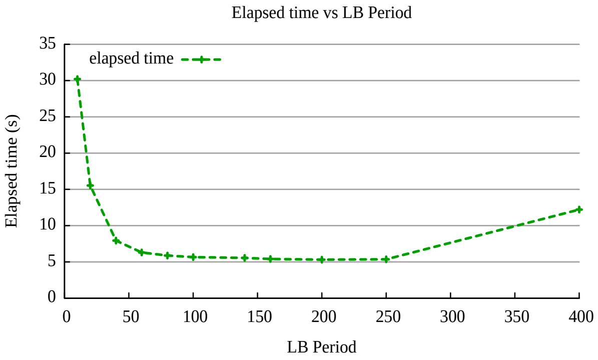
Ideal LB Period

Assuming,
τ - ideal LB period, γ - total iterations
Γ - execution time, θ - cost of LB
$y = ax + c_a$ - average load line equation
$y = mx + c_m$ - maximum load w.r.t average load

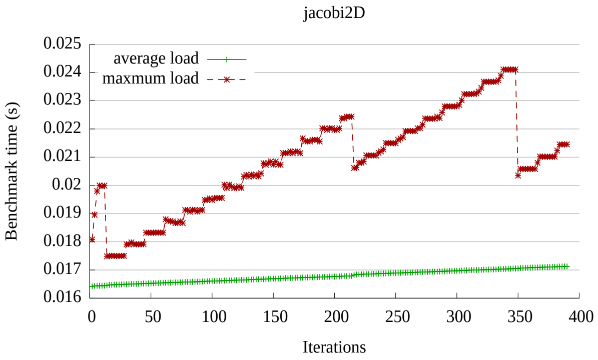
We obtain total execution time as
$\Gamma = \frac{\gamma}{\tau} \times (\int_0^{\tau}(mx + c_m)dx + \theta) + \int_0^{\gamma}(ax + c_a)dx$

Differentiating the above, following LB period is obtained for minimum execution time
$\tau = \sqrt{\frac{2\theta}{m}}$

Results: Jacobi2D



Results: Jacobi2D



LB Period Augmentations

- When the root informs the LB period, some chares may have gone beyond it
- Consensus mechanism to detect such cases, and decide the new LB period

LB Period Augmentations

- When the root informs the LB period, some chares may have gone beyond it
- Consensus mechanism to detect such cases, and decide the new LB period
- As application characteristic changes, LB period may change
    - Capability to refine (expand and contract) LB period if possible

LB Period Augmentations

- When the root informs the LB period, some chares may have gone beyond it
- Consensus mechanism to detect such cases, and decide the new LB period
- As application characteristic changes, LB period may change
  - Capability to refine (expand and contract) LB period if possible
- If prediction and statistics collected do not match, immediate trigger if required

Communication vs Computation

- Applications can be communication bound, computationally intensive, or a mixture of two

Communication vs Computation

- Applications can be communication bound, computationally intensive, or a mixture of two
- Meta-Balancer uses $\alpha\beta$ cost of an application to identify if it is communication intensive, which consist of two components:

  1. α cost - start up cost of the messages sent

Communication vs Computation

- Applications can be communication bound, computationally intensive, or a mixture of two
- Meta-Balancer uses $\alpha\beta$ cost of an application to identify if it is communication intensive, which consist of two components:

  1. α cost - start up cost of the messages sent
  2. β cost - bandwidth cost of bytes sent

Refine vs Comprehensive

- First time load balancing uses comprehensive load balancers

Refine vs Comprehensive

- First time load balancing uses comprehensive load balancers
- Thereafter, refinement strategies are invoked unless history shows poor quality of refinement based strategies

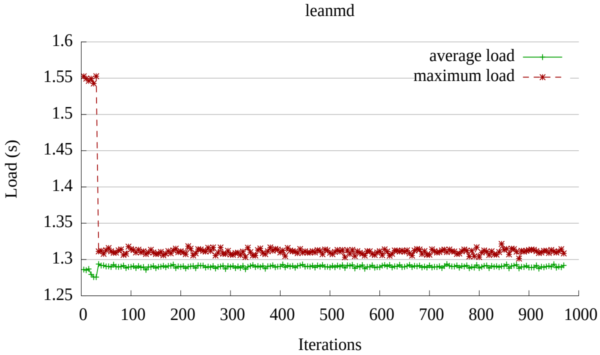


Figure: *leanmd* mini-application

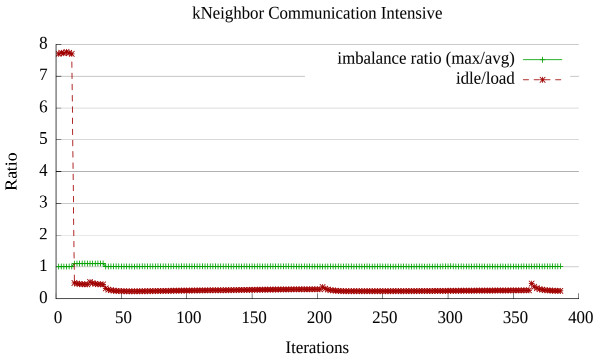


Figure: *kNeighbor* with high communication

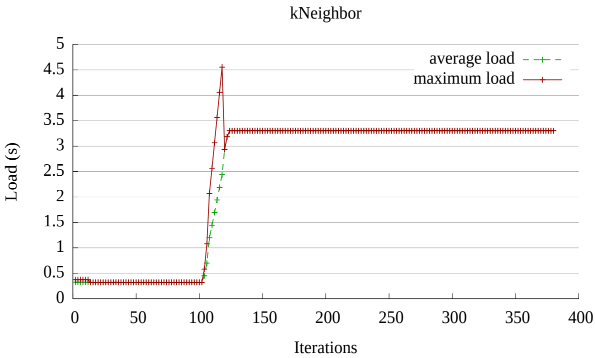
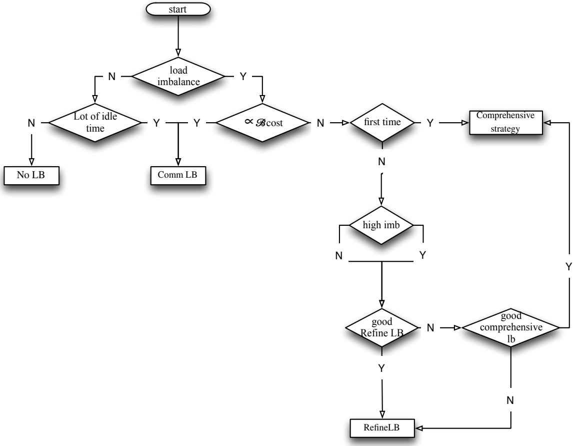


Figure: Dynamic triggering of LB for *kNeighbor*

Overall Scheme

LB Strategy Selection



Outline

1 Motivation

2 Meta-Balancer: Overview

3 Load Balancer: Existing Framework

4 Meta-Balancer
- Statistics Collection
- Ideal LB Period
- Strategy Selection

5 Conclusion

6 Future Work

Conclusion

- Load imbalance affects performance and scalability of an application
- Leaving it to the application programmer to manually handle this imbalance in a dynamic application is unreasonable and inefficient

Conclusion

- Load imbalance affects performance and scalability of an application
- Leaving it to the application programmer to manually handle this imbalance in a dynamic application is unreasonable and inefficient
- Meta-Balancer relieves the user from load balancing decisions by
  - Frequently collecting minimal statistics about the application
  - Controlling the load balancing decision based on the application characteristics

Outline

1 Motivation

2 Meta-Balancer: Overview

3 Load Balancer: Existing Framework

4 Meta-Balancer
- Statistics Collection
- Ideal LB Period
- Strategy Selection

5 Conclusion

6 Future Work

Future Work

- Expand strategy selection
  - Hierarchical vs Centralized
  - Topology-aware vs topology oblivious

Future Work

- Expand strategy selection
  - Hierarchical vs Centralized
  - Topology-aware vs topology oblivious
- More accurate prediction of load - higher order curves

Thank You!