Epidemic Algorithm for Load Balancing

Harshitha Menon, Laxmikant Kalé

15th April

Outline

Outline

Motivation

- Load imbalance in parallel applications
  - Performance is limited by most overloaded processor
  - Leads to drop in system utilization
  - Hampers scalability of the application
  - The chance that one processor is severely overloaded gets higher as no of processors increases
  - For some applications computation load varies over time

Dynamic Load Balancing Framework in Charm++

- Application is composed of large number of migratable units

Dynamic Load Balancing Framework in Charm++

- Application is composed of large number of migratable units
- Load balancing strategy is invoked periodically

Dynamic Load Balancing Framework in Charm++

- Application is composed of large number of migratable units
- Load balancing strategy is invoked periodically
- Based on principle of persistence

Dynamic Load Balancing Framework in Charm++

- Application is composed of large number of migratable units
- Load balancing strategy is invoked periodically
- Based on principle of persistence
- Instruments the application tasks at fine-grained level

Dynamic Load Balancing Framework in Charm++

- Application is composed of large number of migratable units
- Load balancing strategy is invoked periodically
- Based on principle of persistence
- Instruments the application tasks at fine-grained level
- When the load balancing is invoked
  - Gathers the statistics based on the strategy (centralized or hierarchical)
  - Executes load balancing strategy
  - Migrates objects based on new mapping

Load Balancing Strategies

- Centralized Strategies
  - Has global view of the system (good quality load balancing)
  - Clear bottleneck beyond few thousand processors
- Distributed Strategies
  - Processors make autonomous decisions based on local view (neighborhood)
  - Scalable
  - Yield poor load balance due to limited information
- Hierarchical Load balancer
  - Subgroup of processors collect information at the root and receive aggregated information at higher levels
  - Scalable and good quality
  - May suffer from excessive data collection at lowest levels

Outline

Grapevine - Proposed Distributed Load Balancer

Key Features

- Fully distributed scheme

Grapevine - Proposed Distributed Load Balancer

Key Features

- Fully distributed scheme
- Use partial information of the global state of the system

Grapevine - Proposed Distributed Load Balancer

Key Features

- Fully distributed scheme
- Use partial information of the global state of the system
- Propabilistic transfer of load

Grapevine - Proposed Distributed Load Balancer

Key Features

- Fully distributed scheme
- Use partial information of the global state of the system
- Propabilistic transfer of load
- Scalable and good quality

Grapevine - Proposed Distributed Load Balancer

Two Phases

Grapevine - Proposed Distributed Load Balancer

Two Phases

- Information propagation

Grapevine - Proposed Distributed Load Balancer

Two Phases

- Information propagation
- Load transfer

Information Propagation

- Based on *gossip* protocol

Information Propagation

- Based on *gossip* protocol
- Each underloaded processor starts the gossip
- Randomly sample peers and send its load information

Information Propagation

- Based on *gossip* protocol
- Each underloaded processor starts the gossip
- Randomly sample peers and send its load information
- On receiving load information,
  - Combine the information with already known
  - Forward it to random peers

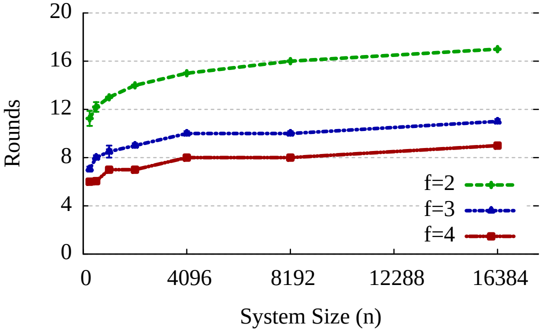
Information Propagation

- Based on *gossip* protocol
- Each underloaded processor starts the gossip
- Randomly sample peers and send its load information
- On receiving load information,
  - Combine the information with already known
  - Forward it to random peers
- No explicit synchronization

Information Propagation

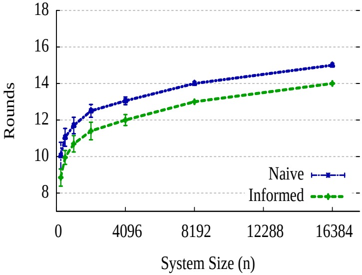
- Number of rounds taken to propagate a single update

$$r = O(\log_f n)$$



Expected number of rounds taken to spread information

Information Propagation



Expected number of rounds taken to spread information

Two Flavors

- Naive
  - Random selection
- Informed
  - Biased selection
  - Incorporate current knowledge

Load Transfer

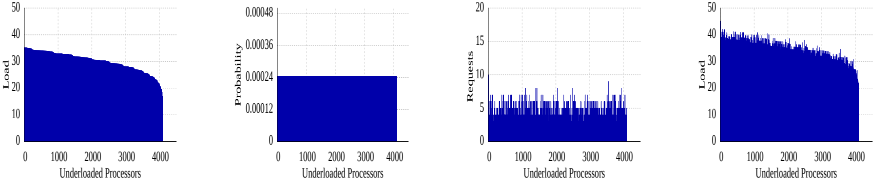
- Probabilistic transfer of load
  - Naive transfer: Select processors uniformly at random
  - Informed transfer: Select processors based on their load

$$p_i = \frac{1}{Z} \times \left(1 - \frac{L_i}{L_{avg}}\right)$$

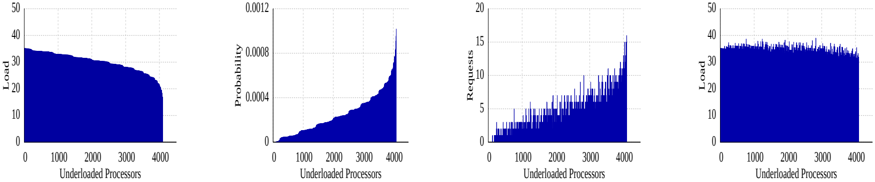
p_i probability assigned to ith processor
L_i load of ith processor
L_{avg} average load of the system
Z normalization constant

Load Transfer

Naive Transfer

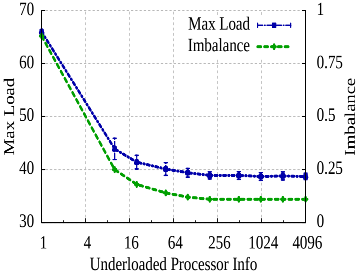


Informed Transfer



(a) Initial load (b) Probabilities assigned (c) Work units transferred (d) Final load.

Quality of Load Balancing



Evaluation of partial information

- Quality is evaluation based on Imbalance given by

$$\mathcal{I} = \frac{L_{max}}{L_{avg}} - 1$$

Outline

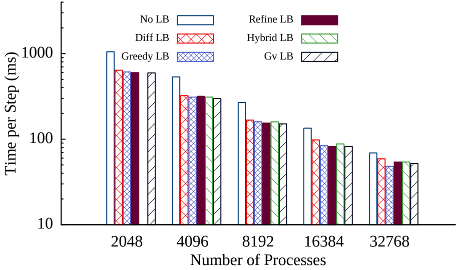
Evaluation

- Applications
  - LeanMD
  - AMR
- Applications were run on IBM BG/Q Vesta
- Comparison with
  - GreedyLB
  - RefineLB
  - AmrLB
  - DiffusionLB
  - HybridLB
- Metrics to evaluate
  - Execution time per step excluding LB time
  - Load balancing overhead
  - Total application time

Evaluation with LeanMD

Time per step

- Quality of our strategy is equivalent to centralized



Evaluation with LeanMD

Load Balancing overhead

- Centralized have high overhead
- Distributed schemes have low overhead

Strategies	Number of Processes				
	2048	**4096**	**8192**	**16384**	**32768**
HybridLB	-	1.35	0.7	0.368	0.2375
GreedyLB	8.62	8.9	10.33	11.2	23.4
RefineLB	55	50	27	34	121
DiffLB	0.039	0.043	0.040	0.043	0.040
GvLB	0.013	0.016	0.023	0.030	0.045

Load balancing cost (in seconds) of various strategies for LeanMD

Evaluation LeanMD

Total application time

- Using centralized strategies overhead exceeds benefit
- Grapevine gives the best performance

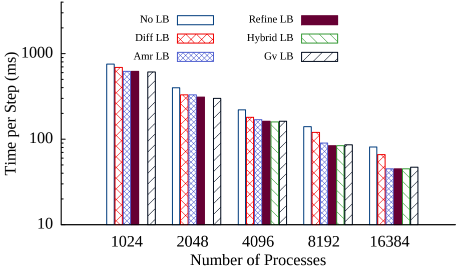
Strategies	Number of Processes				
	2048	**4096**	**8192**	**16384**	**32768**
NoLB	201	102	51	25	13
HybridLB	-	72	37	20	12
GreedyLB	201	148	133	127	243
RefineLB	675	567	306	362	1227
DiffLB	140	72	37	22	13
GvLB	119	64	32	17	10

Total application time (in seconds) for LeanMD on BG/Q

Evaluation with AMR

Time per step

- Quality of our strategy is equivalent to centralized



Evaluation with AMR

Load Balancing overhead

- Centralized have high overhead
- Distributed schemes have low overhead

Strategies	Number of Processes				
	1024	**2048**	**4096**	**8192**	**16384**
HybridLB	-	-	8.29	7.2	2.6
AmrLB	1.09	1.37	2.00	3.30	4.40
RefineLB	12	21	23	33	76
DiffLB	0.015	0.014	0.014	0.014	0.015
GvLB	0.011	0.011	0.015	0.021	0.030

Load balancing cost (in seconds) of various strategies for AMR.

Evaluation with AMR

Total application time

- Load balancing overhead exceeds benefit for most strategies
- Diffusion based load balancer gives marginal benefit
- Grapevine gives the best performance

Strategies	Number of Processes				
	1024	**2048**	**4096**	**8192**	**16384**
NoLB	137	75	43	27	20
HybridLB	-	-	93	69	39
AmrLB	136	69	45	49	47
RefineLB	199	217	209	255	546
DiffLB	135	68	38	25	18
GvLB	123	59	30	21	14

Total application time (in seconds) for AMR on BG/Q.

Outline

Conclusion

- Simple strategy
- Scales well
- Can be tuned to optimize for either cost or quality