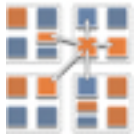
Meta-Balancer: Automated Selection of Load Balancing Strategies

Presenter: Kavitha Chandrasekar

Work done by: *Harshitha Menon, Kavitha Chandrasekar, Laxmikant V. Kale*



PPL UIUC

Outline

Motivation

Need for Dynamic Load-balancing and Strategy Selection

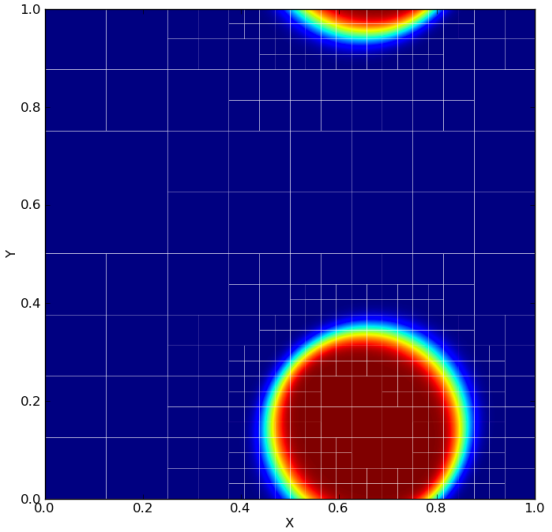
Implementation in Charm++

Meta-Balancer Framework

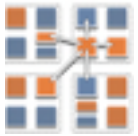
Discussion of application results

Need for Load-balancing

- Several HPC applications exhibit load imbalance, for example:
  - Dynamic computation changes like in AMR
  - Variability in particles per computation unit in Molecular Dynamics
- Load balancing achieves
  - High performance
  - High Resource utilization



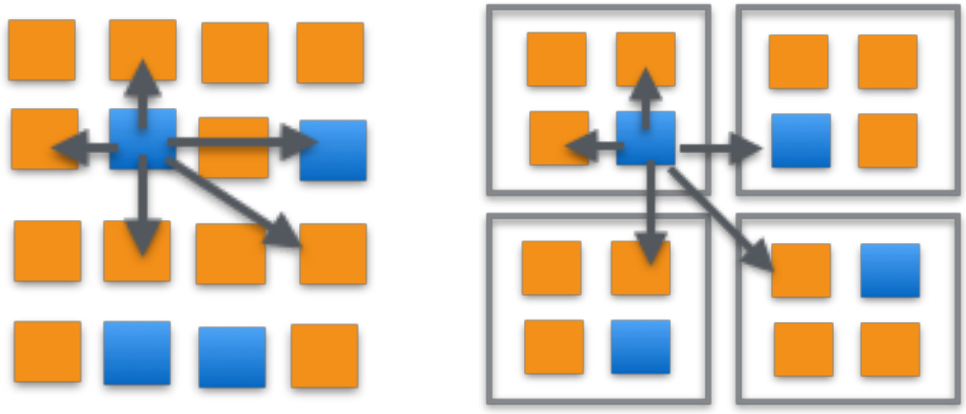
Load Imbalance in AMR application
Reference: AMR mini-app in Charm++



PPL UIUC

Charm++ Programming Model and RTS

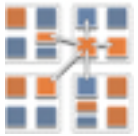
- Asynchronous message-driven execution model
- Supports over-decomposition
  - Dividing computation into more units than hardware processing units
- Supports *Migratability* of work units
- Hence, load balancing is possible
- Supports a suite of load balancing strategies



Over-decomposition of work into Charm++ objects

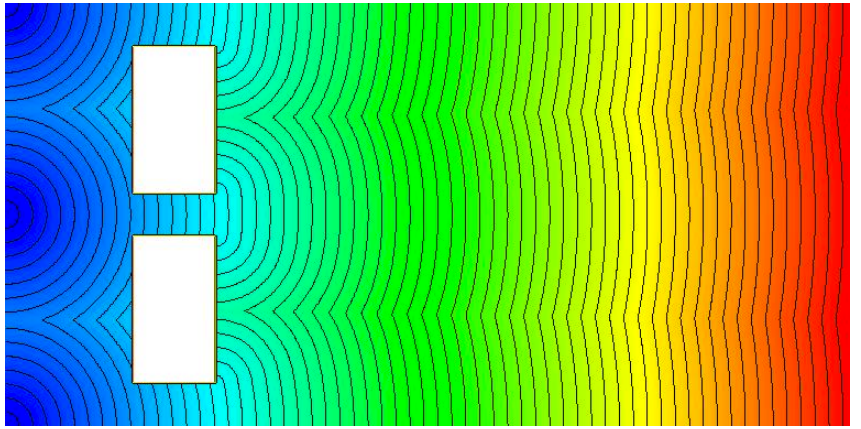
Mapping of Charm++ objects to processors

Charm++ RTS view of over-decomposition



Need for Adaptive Load Balancing Selection

- HPC application have dynamically varying load imbalance
- Applications exhibit varying characteristics
  - Varying over input sizes
  - Varying over phases within an application run
  - Varying core counts
- Load Imbalance characteristics also vary as a result
- *Need for choosing the optimal load balancing strategy*
- Previous work
  - Metabalancer chooses the best load balancing period



Lassen's Propagating Wave Front
Reference: Lassen from LLNL

Differences in Load Balancing Strategies

- Different applications require different load balancing strategies
  - Communication intensive applications require comm-aware load balancers such as graph partitioning-based LBs
  - Computation-based load imbalance may require
    - From-scratch load balancers
    - Refinement-based
- Applications may require different strategies at different stages of the run

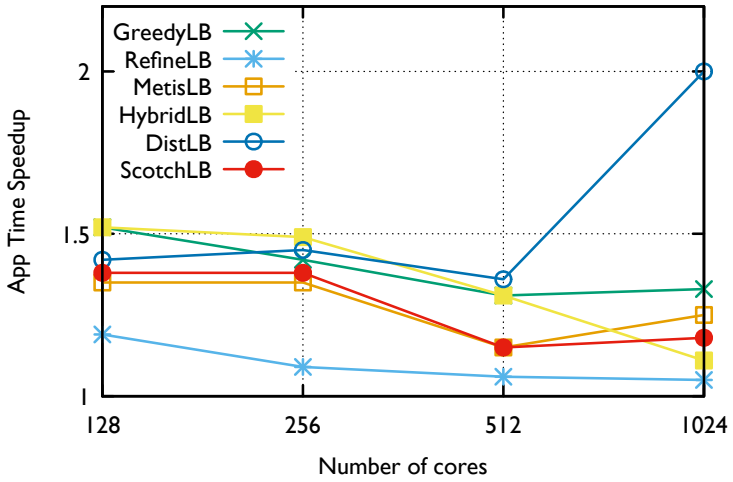
Load balancing Strategies in Charm++

Available load balancing strategies

- Centralized:
  - GreedyLB : From-scratch re-mapping of objects to PEs
  - RefineLB : Re-mapping of objects, taking into account current mapping of objects
  - HierarchicalLB : HybridLB performs GreedyLB at sub-trees and RefineLB at higher-levels
- Distributed:
  - DistributedLB : A distributed strategy for load statistics propagation for mapping of chares from overloaded to underloaded processors
- Communication-aware:
  - MetisLB : Objects are mapped based on partitioning of communication graph
  - ScotchLB : Communication graph partitioning, also taking into account load imbalance
  - ScotchRefineLB : Similar to ScotchLB but takes existing mapping of chares into account

Performance of Load-balancing strategies

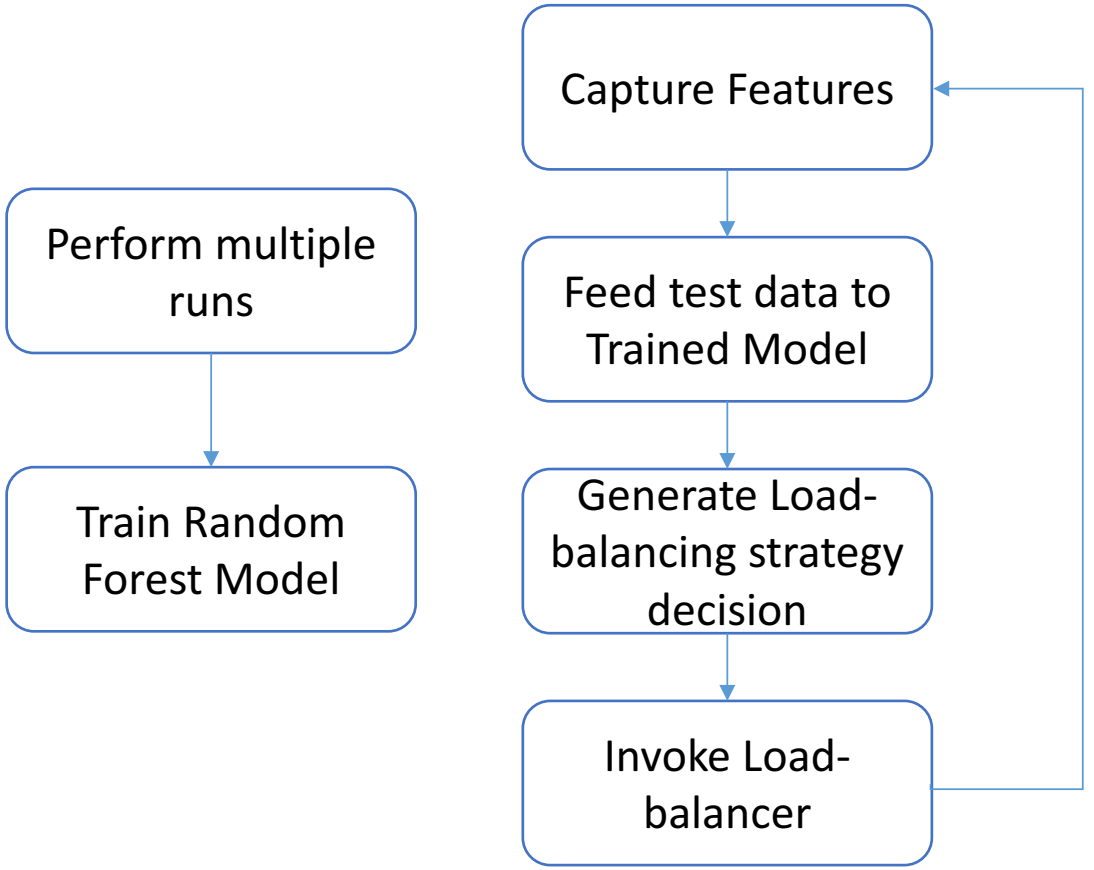
- Different strategies perform differently with different applications, different datasets, varying core counts etc.
- Figure shows performance of Lassen on varying number of cores
- HybridLB shows better speedup for smaller number of cores
- DistributedLB performs better for large core counts
- Even for the same application with weak-scaling, we observe variations in performance

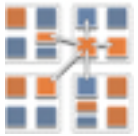


Performance of Lassen with different LBs

Meta-Balancer: For Automated Strategy Selection

- How do we select a strategy from the set of strategies?
- Train a Random Forest Machine learning model offline with training data
- At application runtime, capture application characteristics
- Adaptive run-time system component
  - Identifies when to balance load and which strategy to use
  - Invokes LB and continues to monitor application load imbalance characteristics





Meta-Balancer – Statistics collection

- Objects deposit load information at the iteration boundary
- Aggregation of object load statistics at PEs
  - *AtSync* application calls are used to collect statistics on each PE
  - There is no waiting on barrier unlike regular load balancing at *AtSync*
  - Hence they are very low overhead *AtSync* calls
- Tree based reductions to collect statistics
  - Double array with about 30 features are aggregated by reduction

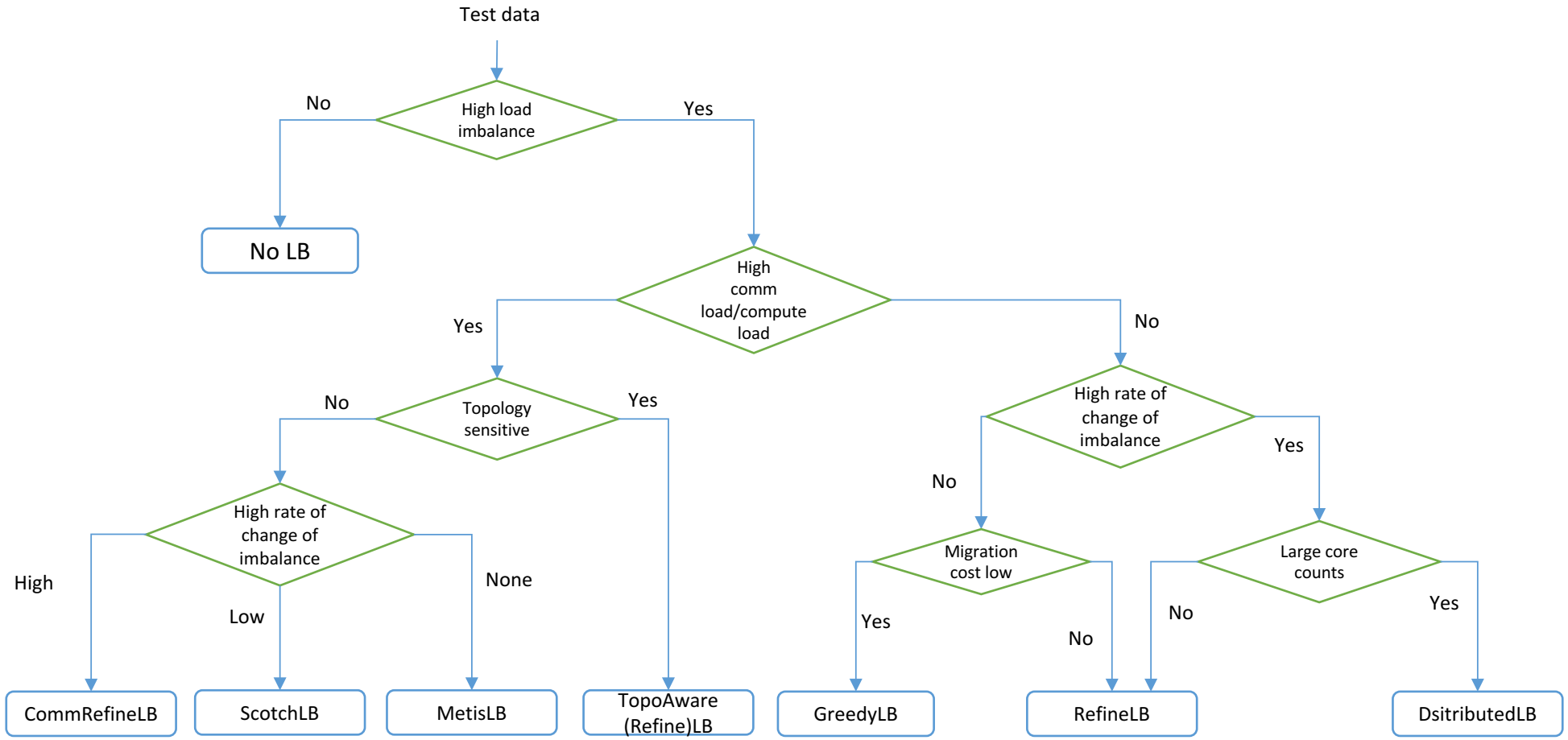
Random Forest Machine Learning technique

- Random forest ML technique to predict the best load balancing strategy - classification task
- We use 100 trees for good prediction accuracy
- Nodes in the tree represent features with associated threshold values
- Leaf Nodes are load-balancing strategy classes with assigned probabilities
- Tunable model parameters for accuracy
  - Tree depth
  - Types of classifier etc.

Random Forest Machine Learning technique

- Why not use a decision tree?
  - Issues in coming up with the right threshold per internal node
  - They can result in overfitting
- We use a decision tree to:
  - Provide a general idea of different scenarios applicable to different LB strategies
  - Decide on Features that can be used to train a model
- Random Forest Model
  - Uses several trees to determine the LB with highest probability
  - Trees are constructed by considering random sets of features

Decision Tree – How strategies vary with differing load imbalance characteristics



Statistics Collected

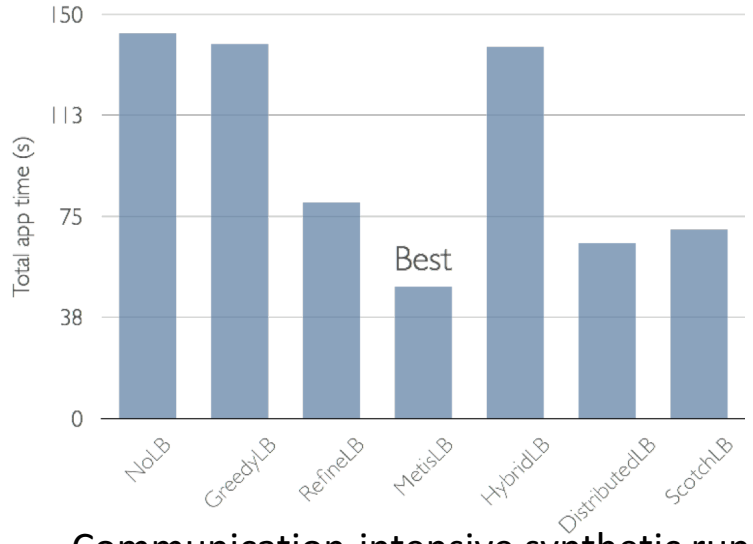
- PE stats
  - Number of PEs
  - PE load imbalance
  - PE utilization (min, max, avg)
- Obj stats
  - Num objs
  - Per PE Obj load (min, max, avg)
- Migration overhead
- Communication stats
  - Total messages, Total bytes, External messages fraction, External bytes fraction, Avg hops, Avg hop bytes, Avg comm neighbors
- Machine stats
  - Latency, Bandwidth
  - Comm cost by Compute cost
- Overhead over Benefit of Centralized strategy

Features generated from Statistics

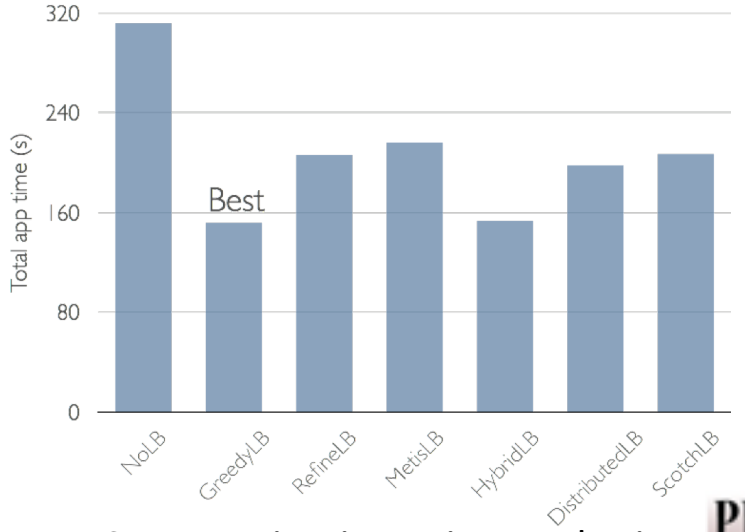
- PE stats
  - PE load imbalance = PE max load / PE avg load
  - Relative rate of change of load (max load, avg load)
- Communication cost by Compute cost
  - Communication cost based on latency, bandwidth
  - Computation cost is the sum of time taken to execute work
- Overhead by Benefit of Centralized strategy
  - Overhead of centralized GreedyLB strategy is O(num_objs * log (p))
  - Benefit of the centralized strategy is based on the assumption that it achieves perfect balance

Training Set

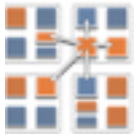
- Synthetic benchmark *lb_test* to study different behavior
  - varying communication pattern
  - computation load
  - migration cost
  - message sizes
- These parameters allow us to introduce application characteristics
  - Load imbalance
  - Dynamic load imbalance
  - Compute vs Communication-intensive



Communication-intensive synthetic run

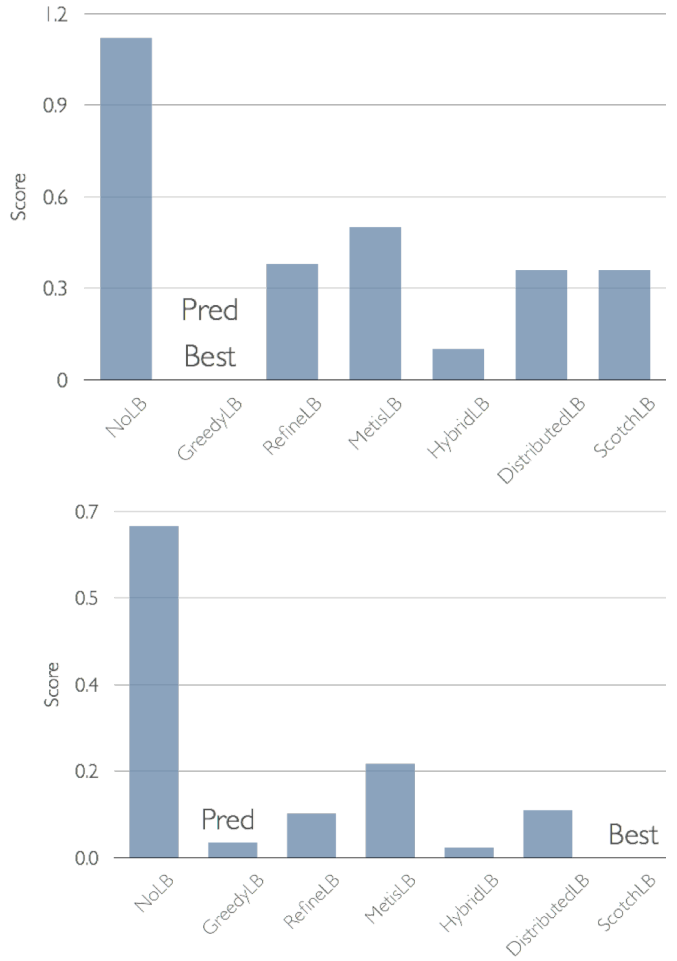


Computation-intensive synthetic run



Predictions on Test Data

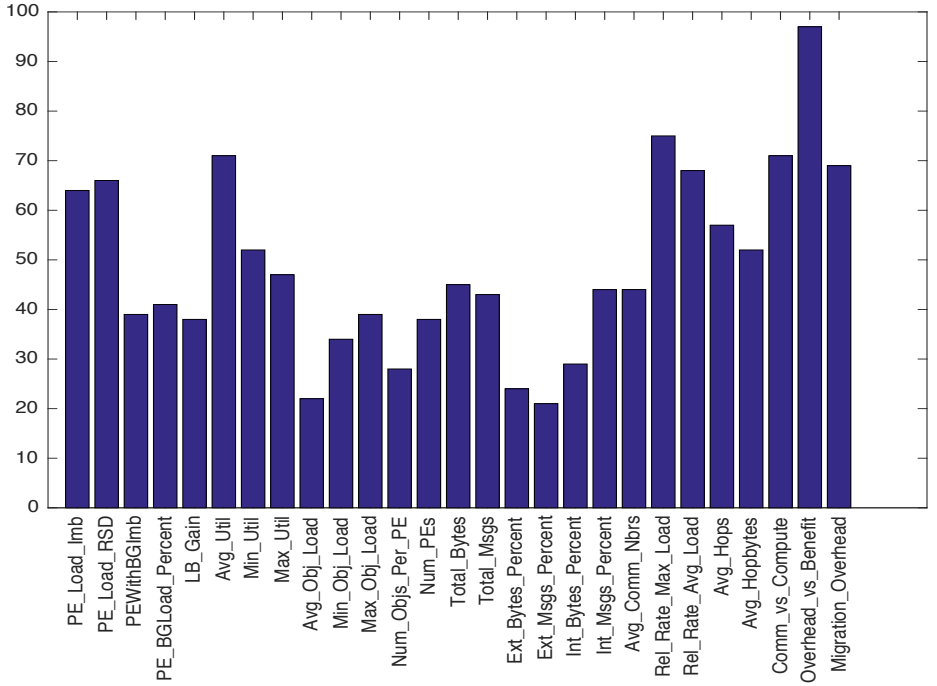
- 75 configurations were used in training set and 25 in test set
- Variations in parameters allowed to simulate different application characteristics
  - Eg: Communication-intensive, Compute-intensive
- score = $T_{predicted\_lb} / T_{best\_lb} - 1$
- Larger the score, worse the performance



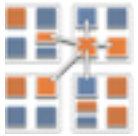
Prediction Scores

Histogram of Features

- Histogram of features used for sample test data
- 100 trees are traversed by the classifier to find the load balancer with maximum probability
- We output how often a feature is used to make a decision on the path
- Some key features were:
  - PE_load_Imb
  - Avg_utilization
  - Overhead_vs_benefit
  - Comm_vs_Compute



Histogram of features used by Random Forest for synthetic test data



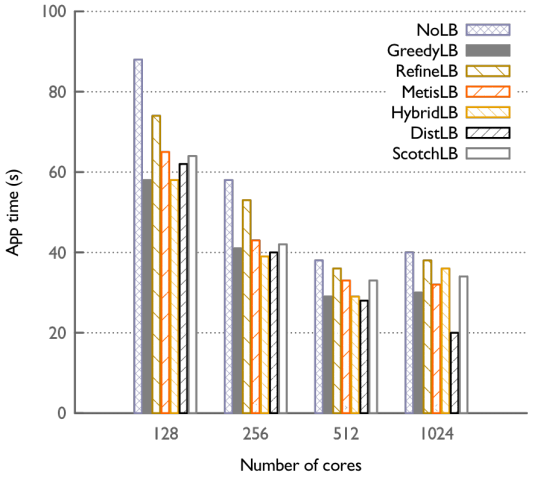
PPL UIUC

Using Trained Model for applications

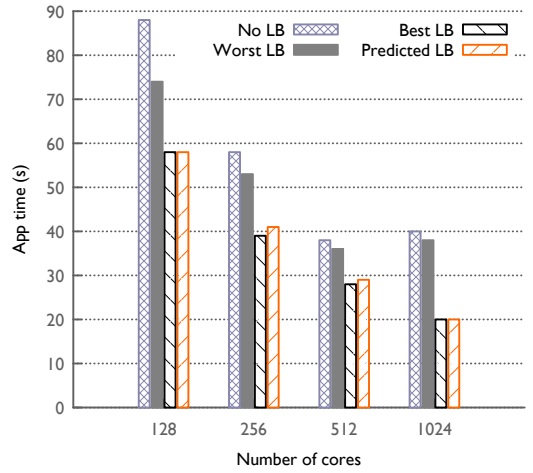
- Trained Random Forest Model from synthetic runs was used with
  - Lassen
  - PRK Particle-In-Cell Simulation
  - NPB BT-MZ
  - PDES
  - LeanMD
- Some of the mini-application results are presented here

Lassen – Wave propagation application

- Used to study detonation shock dynamics
- Suffers from varying load imbalance
- Meta-Balancer is able to choose a good load balancer
  - In most cases, atleast the 2nd best load balancer
- Meta-Balancer predicts LB (DistLB - 1024 cores) that improves performance by 2X for worst LB



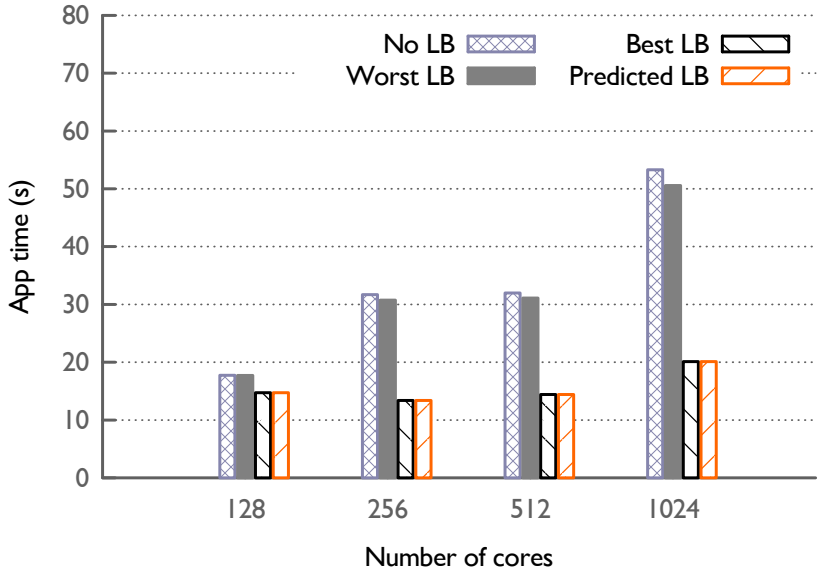
Application times for all LBs



Lassen Prediction for LB strategies

Prediction for Particle-In-Cell Simulation

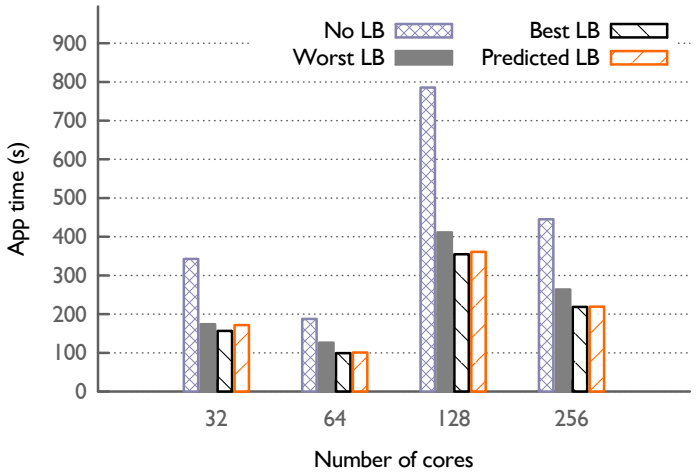
- Charm++ implementation of PRK PIC
- Used for simulation of plasma particles
- Load imbalance results from imbalanced distribution of particles and particle motion
- Meta-Balancer is able to predict GreedyLB that improves performance by more than 2X for worst LB



PRK PIC Prediction for LB strategies

Prediction for NPB BT-MZ

- Use of Random Forest Algorithm for prediction of Load balancer
- Example: NPB BT-MZ, an AMPI benchmark
- Uneven zone size – load imbalance
- Communication intensive
- Input classes used: C and D
- For core 128, 256 cores, prediction is the second best



NPB BT-MZ Prediction for LB strategies

Prediction for NPB BT-MZ

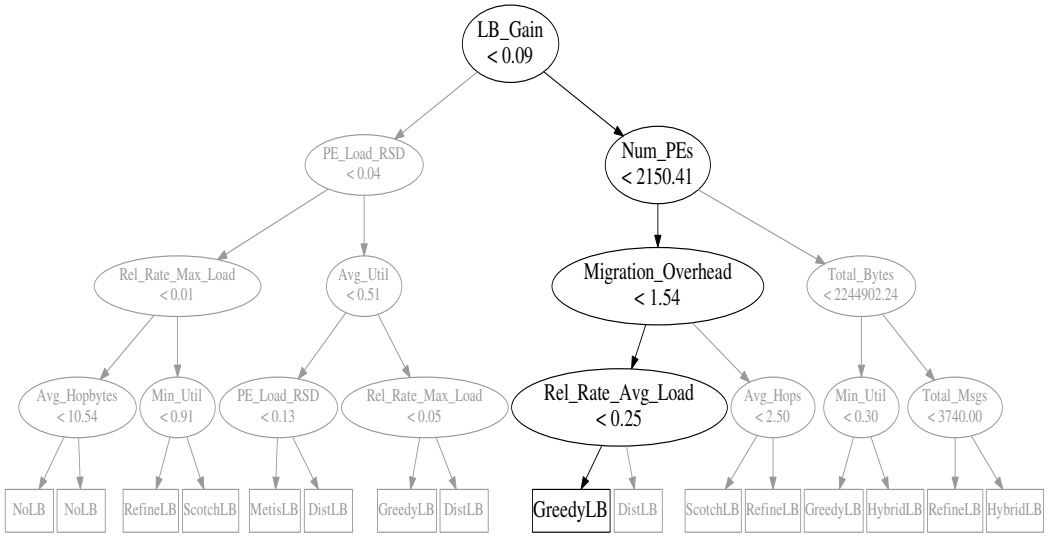
Cores	NoLB	GreedyLB	RefineLB	MetisLB	ScotchLB	HybridLB	DistributedLB	Predicted LB
32	0.46	0.92	0.97	0.90	0.91	1.00	0.90	0.91 (ScotchLB)
64	0.53	0.99	0.92	1.00	0.98	0.93	0.79	0.98 (ScotchLB)
128	0.45	0.98	0.98	0.86	1.00	0.98	0.92	0.98 (GreedyLB)
256	0.49	1.00	0.95	0.83	0.99	0.95	0.87	0.99 (ScotchLB)

Load-Balancing Score for NPB BT-MZ

- Score of prediction with respect to performance speedup is shown
- For core 128, 256 cores, prediction is the second best
- Smaller cores show might have variability due to short run time

Tree Path description for BT-MZ

- Result Analysis:
  - What features work?
- Figure shows BT-MZ's LB prediction
- LB_Gain (overhead_vs_benefit) is > 0.09
  - Hence centralized strategy is acceptable
- NUM_PEs is low
  - Hence Greedy is acceptable (instead of Hybrid)
- Migration overhead is low
  - Hence Greedy is acceptable than Refine
- Rate of imbalance is low
  - Hence greedy is acceptable, since it can be called infrequently with low overhead



Random Forest's tree for BT-MZ's LB prediction

Summary: Meta-Balancer for Strategy Selection

- Random Forest algorithm for training and to predict suitable load balancing strategy with high accuracy
- Adaptive run-time component to make the load balancing strategy selection decision
- Several applications benefit from strategy selection
- Our Result Analysis looks at why a Load-balancing strategy was predicted with high probability

Thank You