Workload Management for Big Data Analytics

Ashraf Aboulnaga
University of Waterloo +
Qatar Computing Research Institute

Shivnath Babu
Duke University

Database Workloads

	On-line	Batch
Transactional	Airline Reservation	Payroll
Analytical	OLAP	BI Report Generation

This tutorial

- Different tuning for different workloads
- Different systems support different workloads
- Trend towards ***mixed workloads***
- Trend towards ***real time*** (i.e., more on-line)

Big Data Analytics

- Complex analysis (on-line or batch) on
  - Large relational data warehouses +
    Web site access and search logs +
    Text corpora +
    Web data +
    Social network data +
    Sensor data +
    …etc.

- Different systems with different data types, programming models, performance profiles, … etc.

Big Data Analytics Ecosystem

- Rich set of software systems support Big Data analytics

- Focus of this tutorial:
  - ***Parallel database systems***
  - ***MapReduce (Hadoop) including Pig Latin and Hive***

- Other systems also exist
  - SCOPE – SQL-like language with rich optimization
  - Pregel, GraphLab, PowerGraph – graph processing
  - Spark, HaLoop, HadoopDB – improvements on Hadoop
  - R, Weka, Matlab – traditional statistical analysis

Big Data Analytics Ecosystem

- Diverse set of storage systems
  - Hadoop File System (HDFS)
  - NoSQL systems such as HBase, Cassandra, MongoDB
  - Relational databases

Complex software and infrastructure

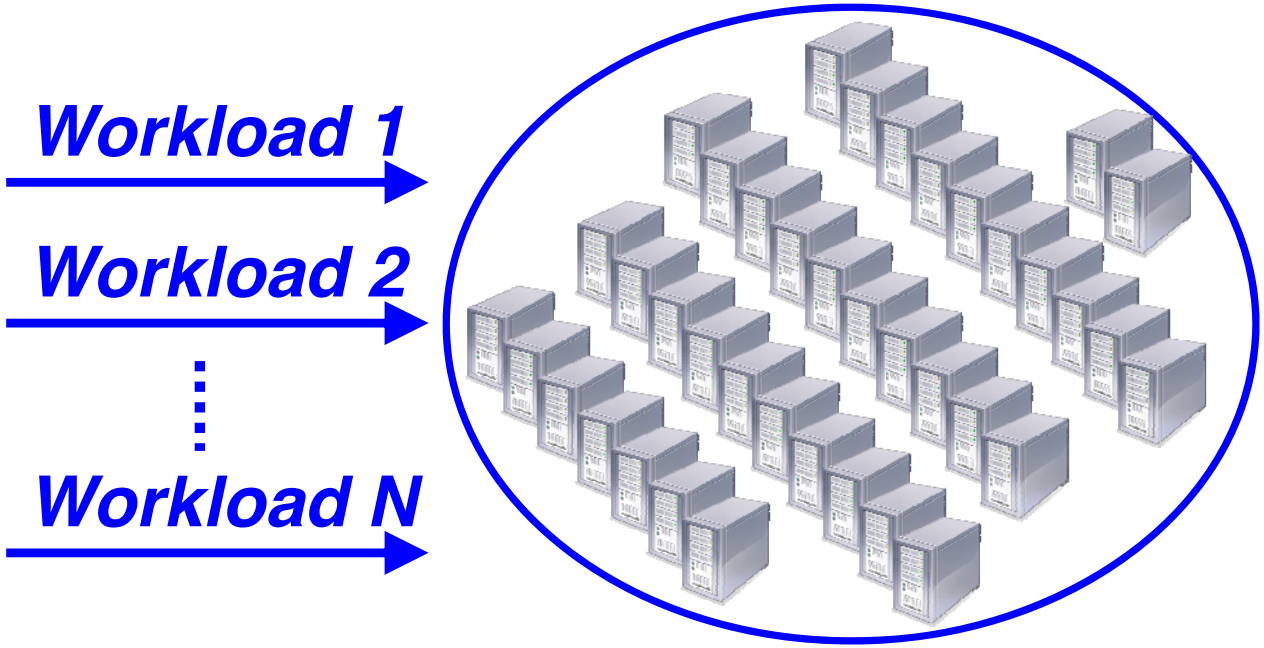
Multiple concurrent workloads

Administration is difficult

Some systems provide mechanisms for administration, but not policies

Need tools to help administrator

Workload Management



- Workloads include all queries/jobs and updates
- Workloads can also include administrative utilities
- Multiple users and applications (multi-tenancy)
- Different requirements
  - Development vs. production
  - Priorities

Workload Management

- Manage the execution of multiple workloads to meet ***explicit or implicit service level objectives***

- ***Look beyond the performance of an individual request to the performance of an entire workload***

Problems Addressed by WLM

- Workload isolation
  - Important for multi-tenant systems
- Priorities
  - How to interpret them?
- Admission control and scheduling
- Execution control
  - Kill, suspend, resume
- Resource allocation
  - Including sharing and throttling
- Monitoring and prediction
- Query characterization and classification
- Service level agreements

Optimizing Cost and SLOs

- When optimizing workload-level performance metrics, balancing ***cost (dollars)*** and ***SLOs*** is always part of the process, whether implicitly or explicitly
- Also need to account for the effects of ***failures***

Run each workload on an independent overprovisioned system

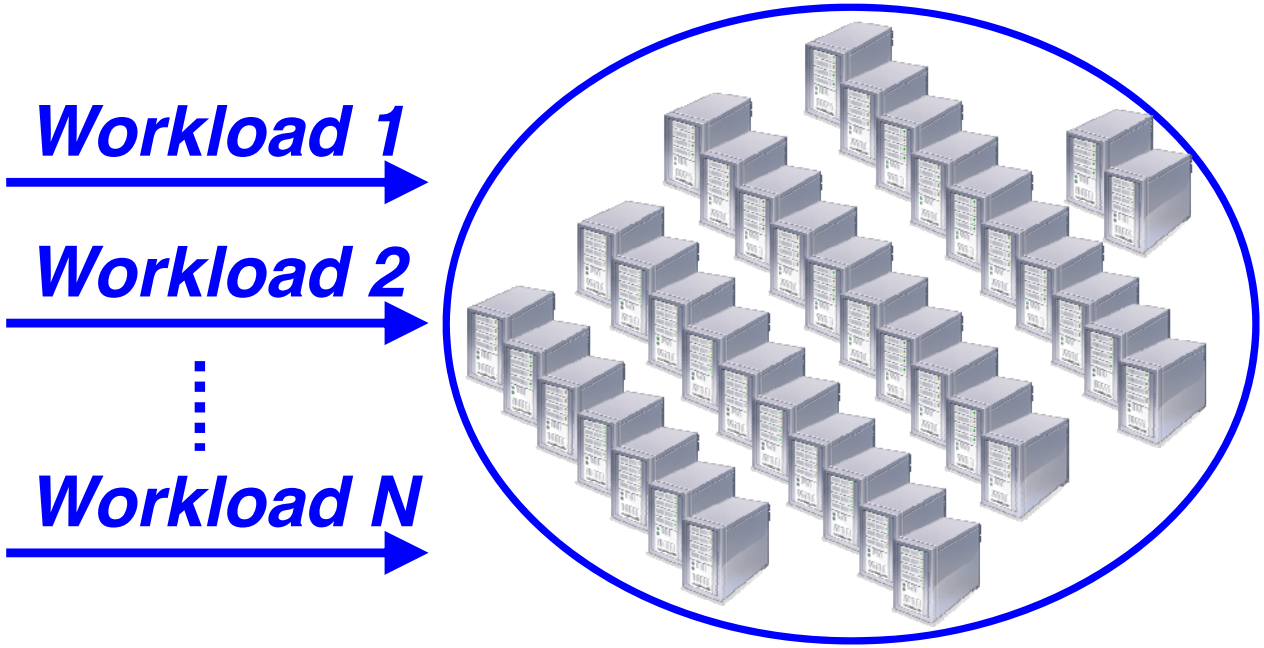
(Cost is not an issue)

Run all workloads together on the smallest possible shared system

(No SLOs)

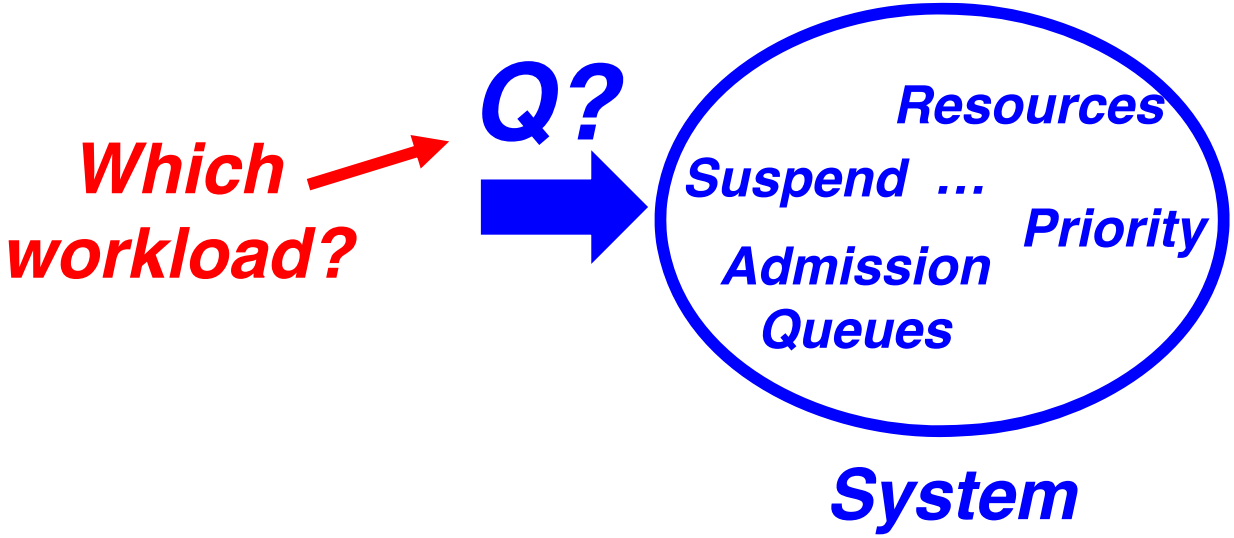
Example: A dedicated business intelligence system with a hot standby

Recap



Workload management is about controlling the execution of different workloads so that they achieve their SLOs while minimizing cost (dollars). Effective workload management is essential given the scale and complexity of the Big Data analytics ecosystem.

Defining Workloads



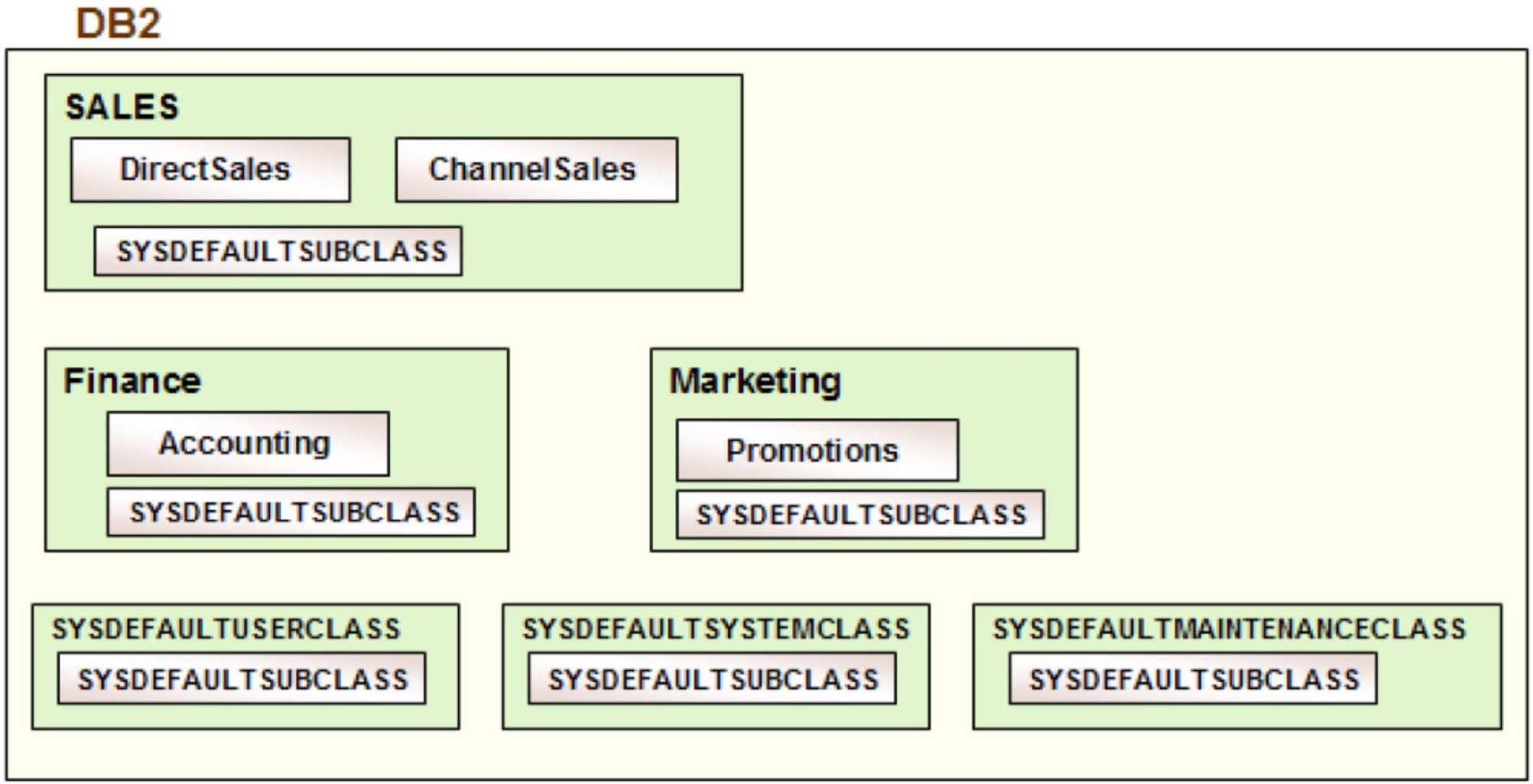
- Specification (by administrator)
  - Define workloads by connection/user/application
- Classification (by system)
  - Long running vs. short
  - Resource intensive vs. not
  - Just started vs. almost done

DB2 Workload *Specifictaion*

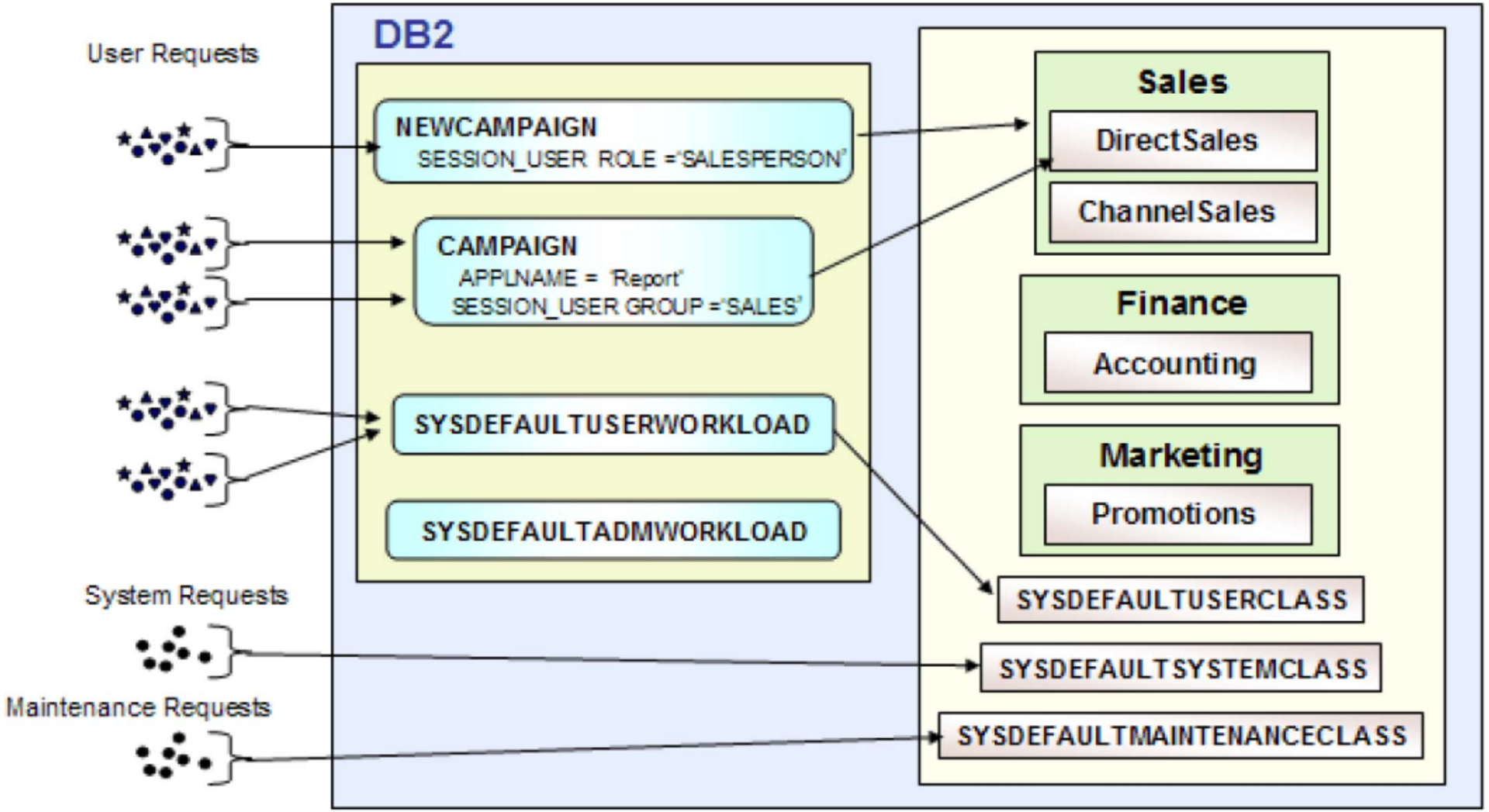
Whei-Jen Chen, Bill Comeau, Tomoko Ichikawa, S Sadish Kumar, Marcia Miskimen, H T Morgan, Larry Pay, Tapio Väättänen. “DB2 Workload Manager for Linux, UNIX, and Windows.” *IBM Redbook*, 2008.

- Create ***service classes***
- Identify workloads by connection
- Assign workloads to service classes
- Set thresholds for service classes
- Specify action when a threshold is crossed
  - Stop execution
  - Collect data

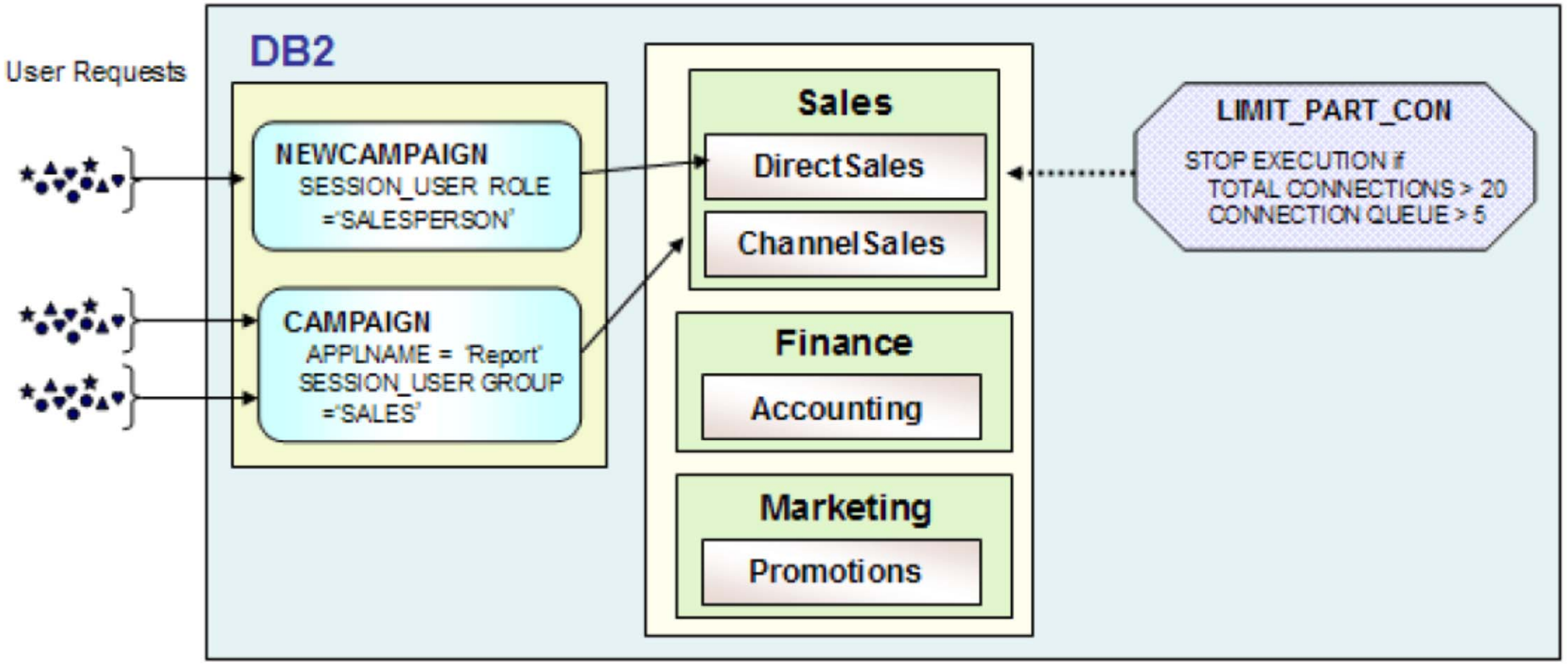
Service Classes in DB2



Workloads in DB2



Thresholds in DB2



Many mechanisms available to the DBA to specify workloads. Need guidance (policy) on how to use these mechanisms.

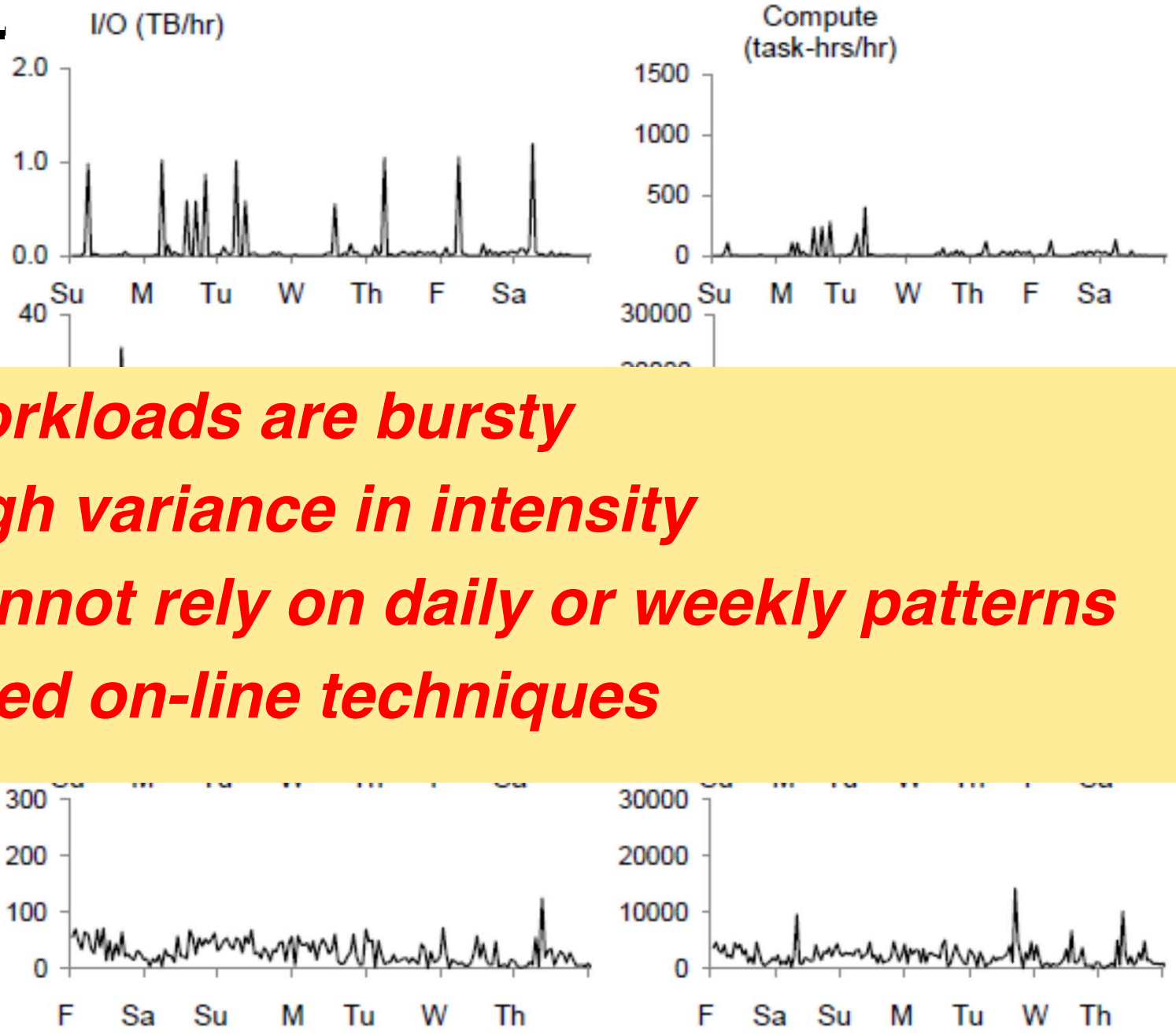
MR Workload *Classification*

Yanpei Chen, Sara Alspaugh, Randy Katz. "Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads." *VLDB*, 2012.

- MapReduce workloads from Cloudera customers and Facebook

Trace	Machines	Length	Date	Jobs	Bytes moved
CC-a	<100	1 month	2011	5759	80 TB
CC-b	300	9 days	2011	22974	600 TB
CC-c	700	1 month	2011	21030	18 PB
CC-d	400-500	2+ months	2011	13283	8 PB
CC-e	100	9 days	2011	10790	590 TB
FB-2009	600	6 months	2009	1129193	9.4 PB
FB-2010	3000	1.5 months	2010	1169184	1.5 EB
Total	>5000	≈ 1 year	-	2372213	1.6 EB

Variation Over Time



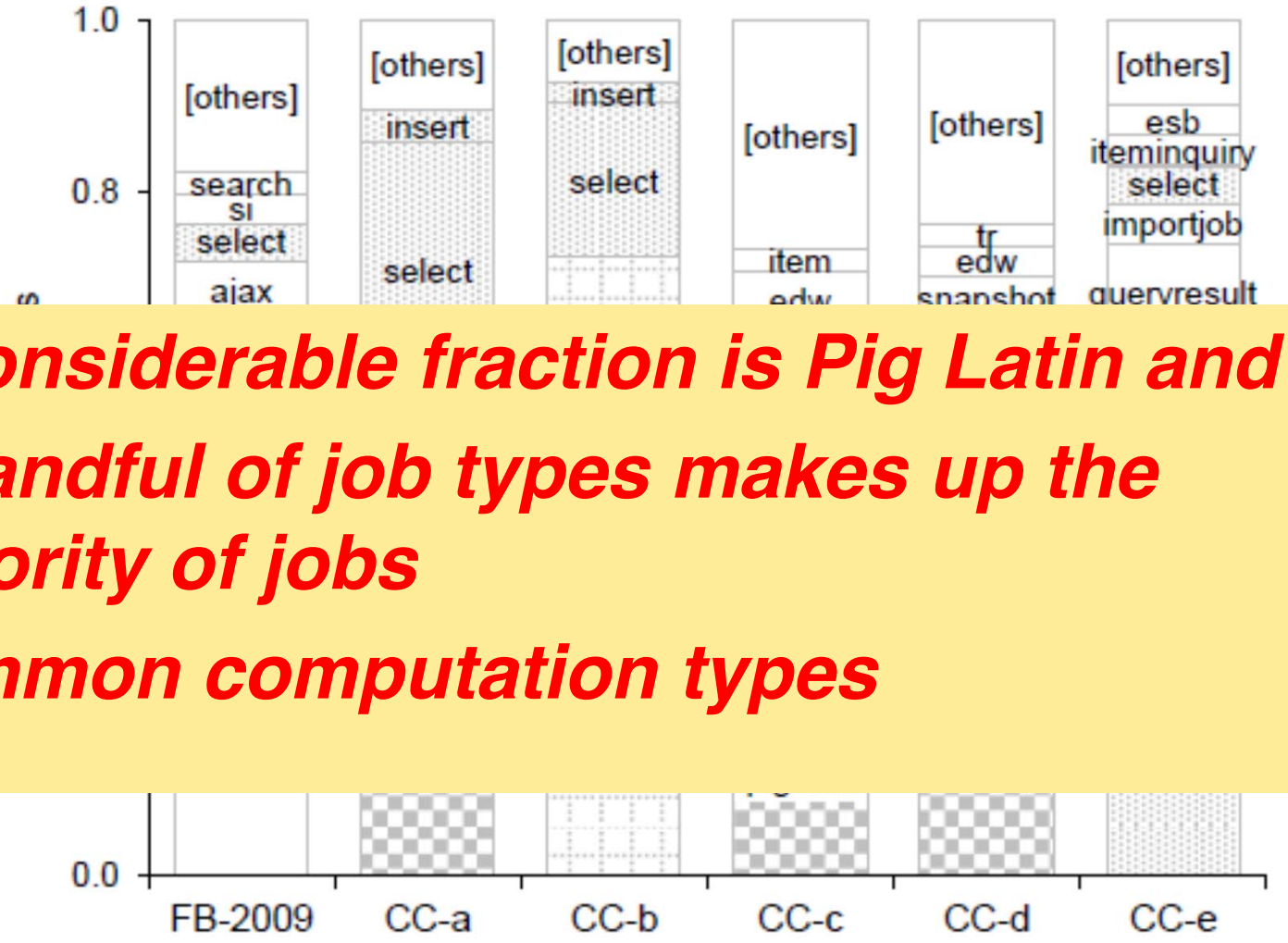
Workloads are bursty

High variance in intensity

Cannot rely on daily or weekly patterns

Need on-line techniques

Job Names



A considerable fraction is Pig Latin and Hive

A handful of job types makes up the majority of jobs

Common computation types

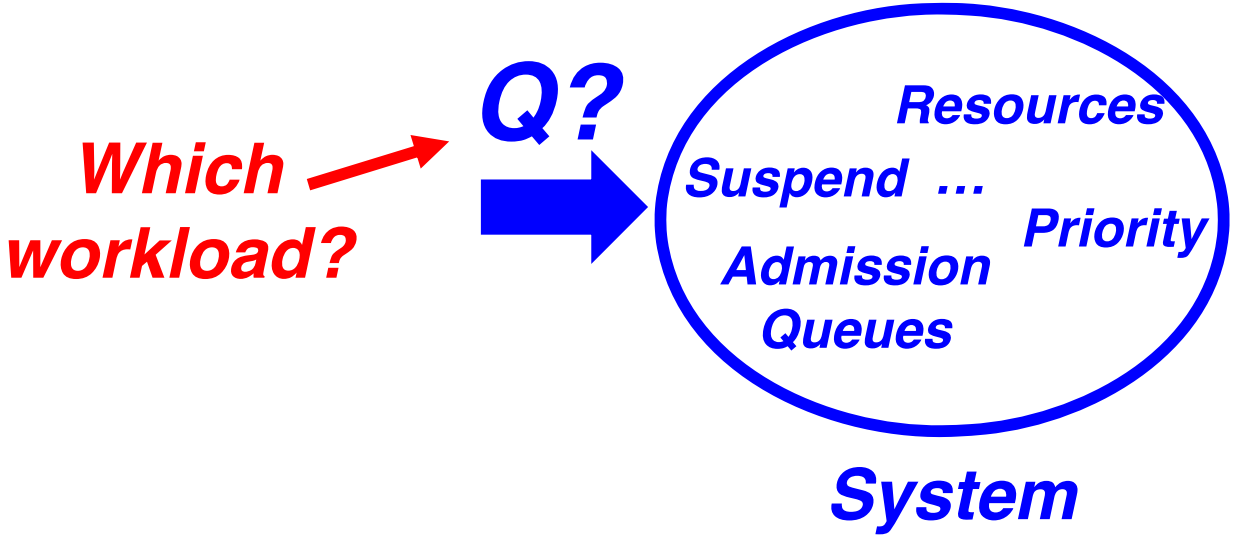
Job Behavior (k-Means)

	# Jobs	Input	Shuffle	Output
CC-a	5525	51 MB	0	3.9 MB
	194	14 GB	12 GB	10 GB
	31	1.2 TB	0	27 GB
	9	273 GB	185 GB	21 MB

Diverse job behaviors

Workloads amenable to classification

Recap



- *Can specify workloads by connection/user/application.*
- *Mechanisms exist for controlling workload execution.*
- *Can classify queries/jobs by behavior.*
- *Diverse behaviors, but classification still useful.*

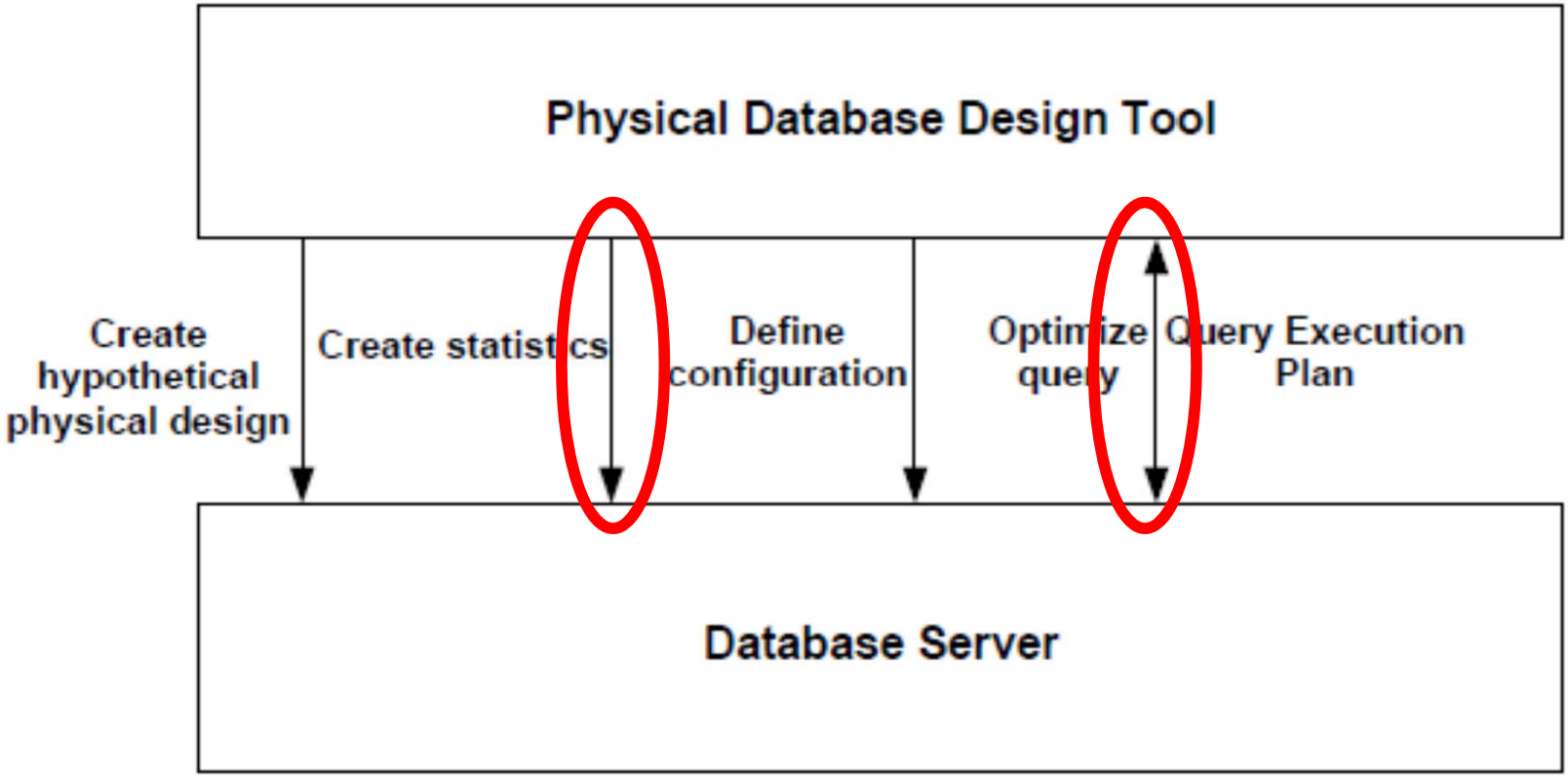
Tutorial Outline

- Introduction
- Workload-level decisions in database systems
  - Physical design, Scheduling, Progress monitoring, Managing long running queries
- Performance prediction
- ***Break***
- Inter-workload interactions
- Outlook and open problems

Workload-level Decisions in Database Systems

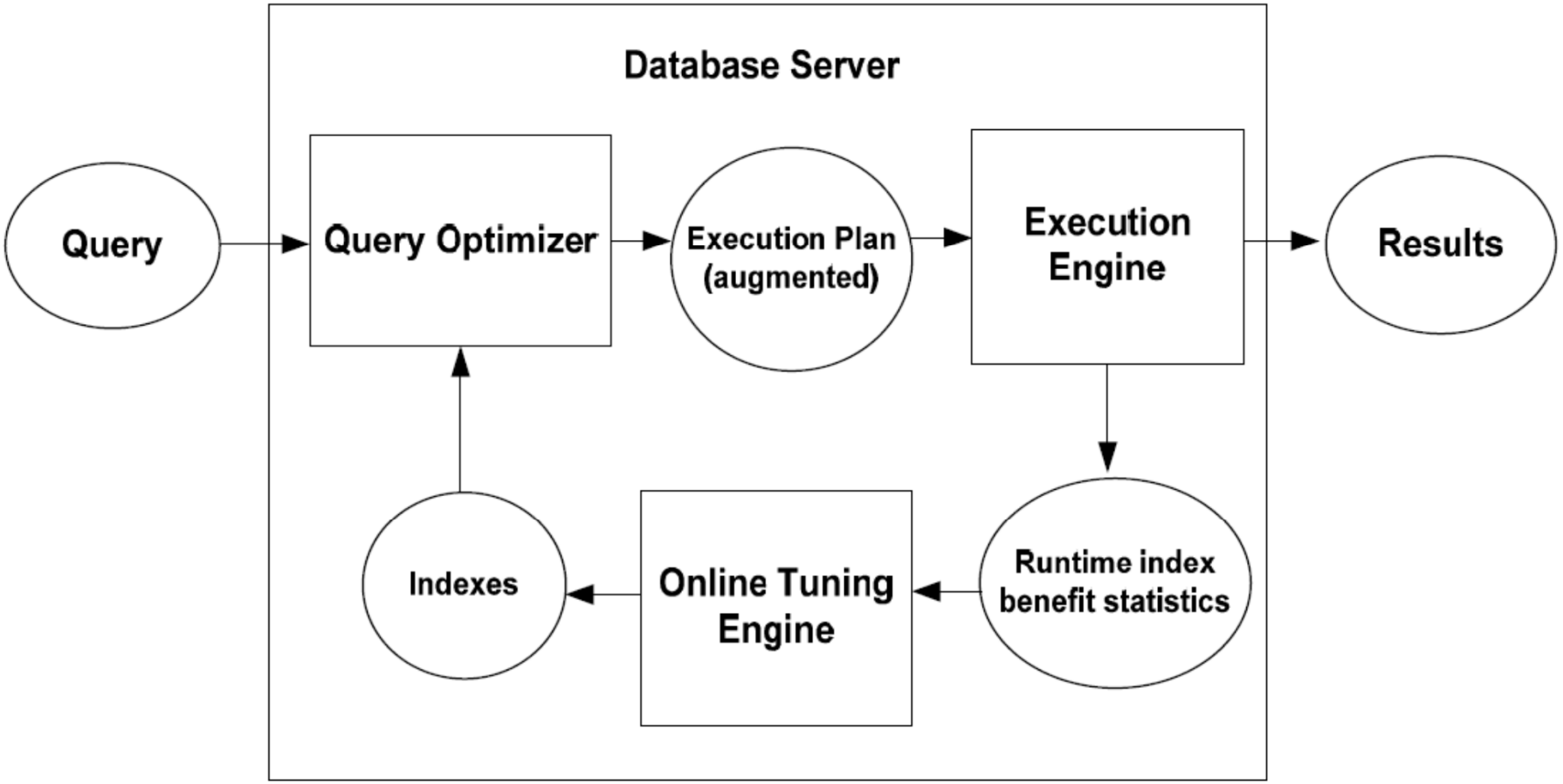
Physical Database Design

Surajit Chaudhuri, Vivek Narasayya. “Self-Tuning Database Systems: A Decade of Progress.” *VLDB*, 2007.



- A workload-level decision
- Estimating benefit relies on query optimizer

On-line Physical Design

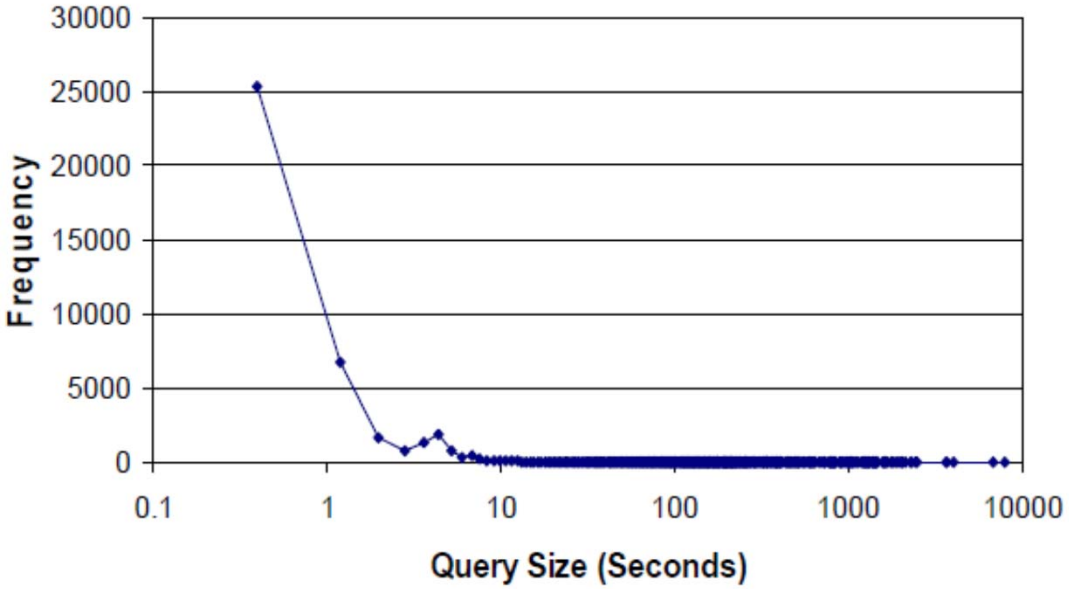


- Adapts the physical design as the behavior of the workload changes

Scheduling BI Queries

Chetan Gupta, Abhay Mehta, Song Wang, Umeshwar Dayal. "Fair, Effective, Efficient and Differentiated Scheduling in an Enterprise Data Warehouse." *EDBT,* 2009.

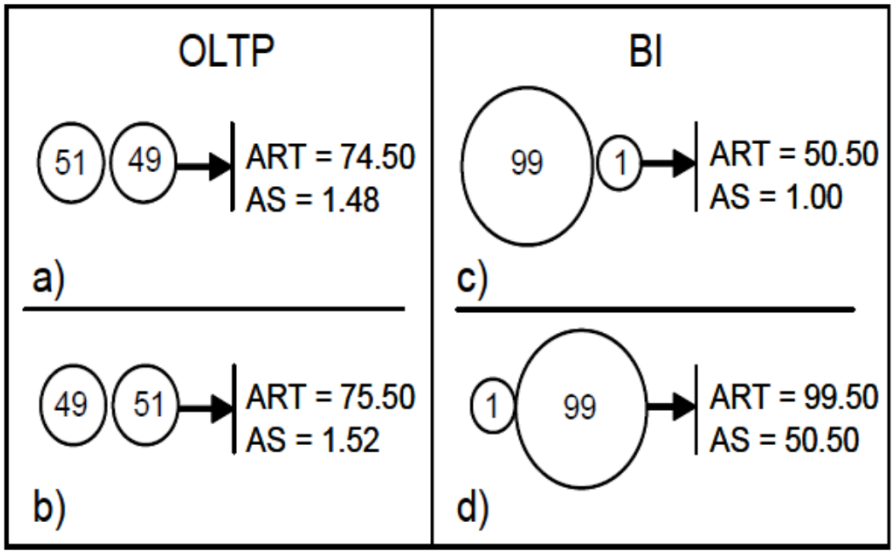
- High variance in execution times of BI queries
- Challenging to design a ***mixed workload scheduler***



(Distribution of one day of queries from an actual data warehouse)

Stretch Metric for Scheduling

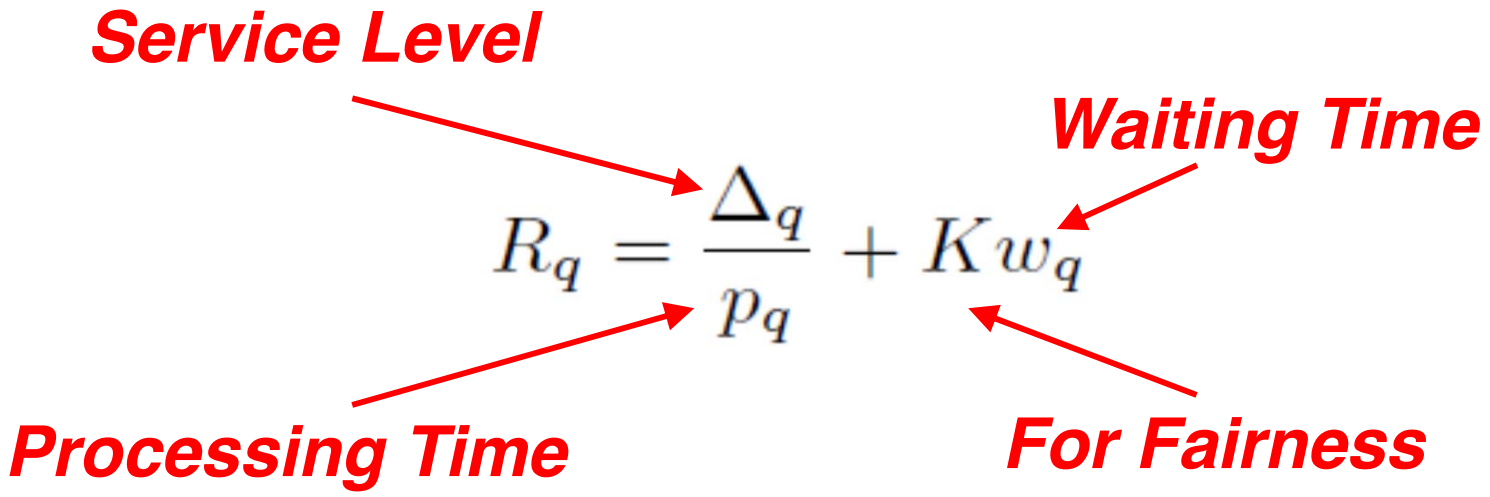
- ***Stretch*** of a query defined as
  (wait time + execution time) / execution time
- Same as response ratio in operating systems scheduling



(ART = Average Response Time, AS = Average Stretch)

rFEED Scheduler

- Fair – Minimize the maximum stretch
  Effective – Minimize the average stretch
  Efficient – Sub-linear implementation
  Differentiated – Different service levels
- Ranking function for queries



(Δ and K depend on maximum processing time and degree of bias towards higher service levels)

Scheduling With Interactions

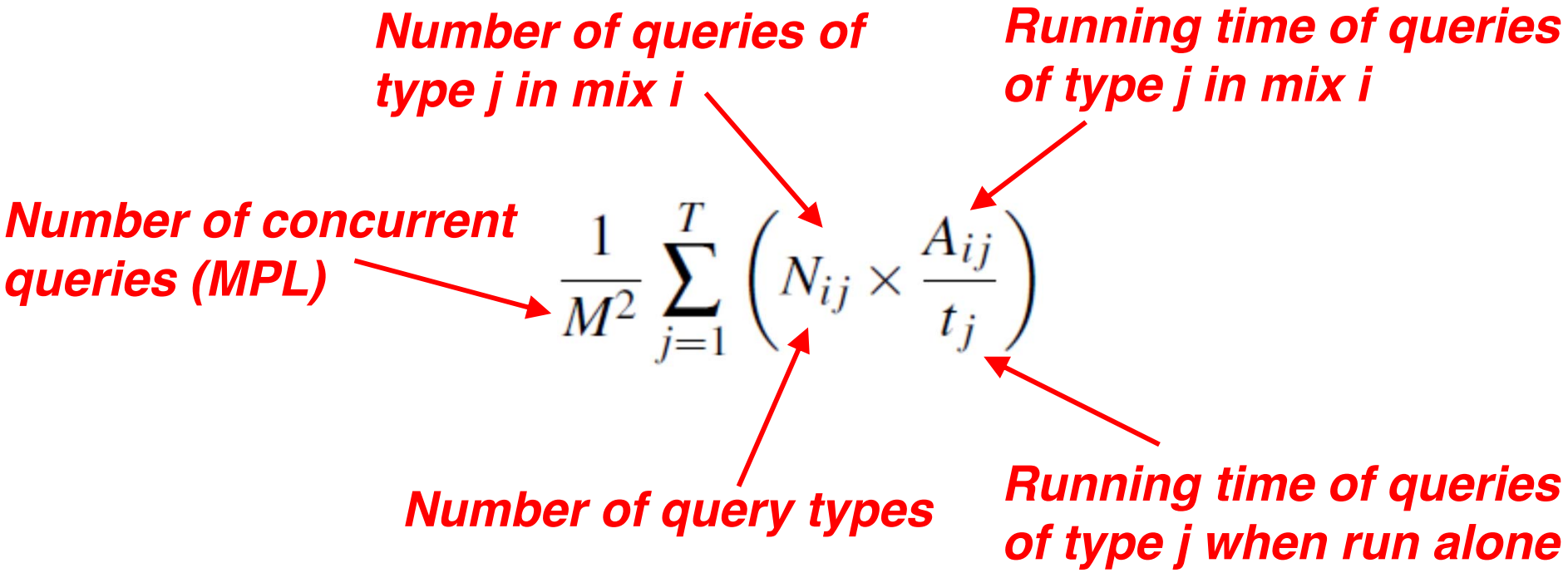
Mumtaz Ahmad, Ashraf Aboulnaga, Shivnath Babu, Kamesh Munagala. "Interaction-Aware Scheduling of Report Generation Workloads." *VLDBJ,* 2011.

- Typical database workload consists of a ***mix of interacting queries***
- How to schedule in the presence of query interactions?

- A ***query mix*** is a set of queries that execute concurrently in the system. Given *T* query types:
  - $m = <N_1, N_2, \ldots, N_T>$, where N_i is the number of queries of type *i* in the mix
- Schedule a ***sequence of query mixes***

NRO Metric for Scheduling

- Normalized Run-time Overhead (NRO)



- Measures overhead added by query interactions
- Keep NRO at a target value (avoid future overload)

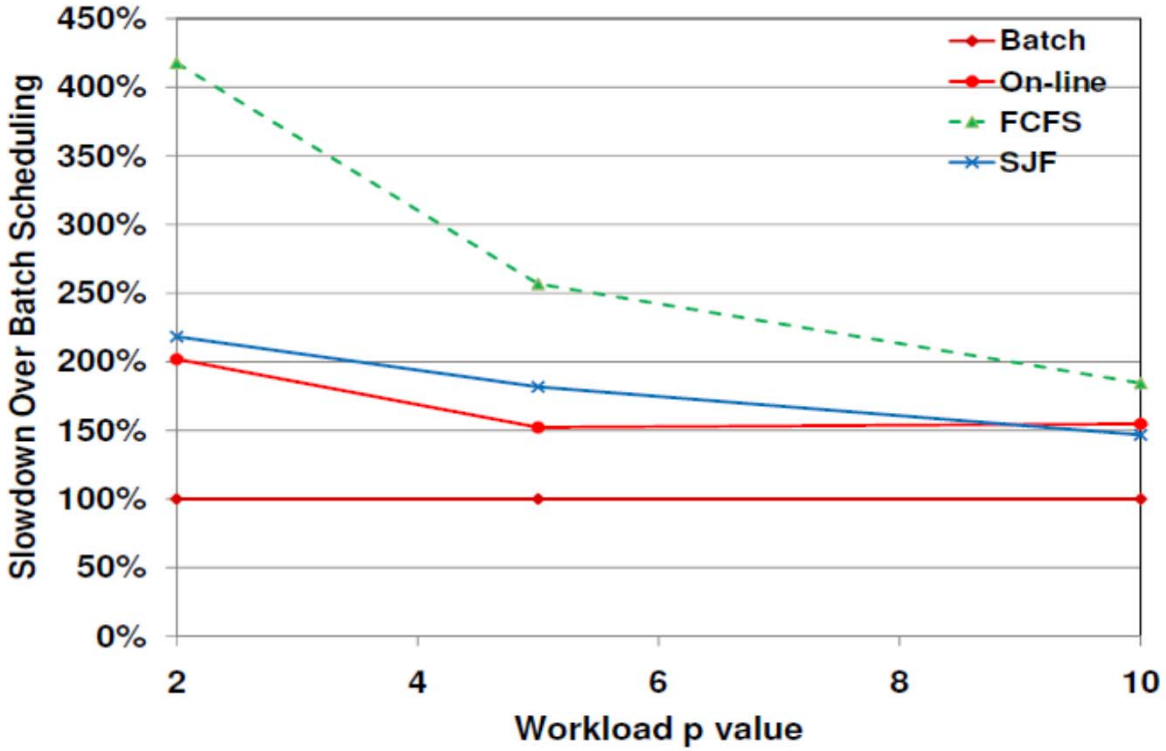
Modeling Query Interactions

- Given a query mix, predict the average ***query completion time (A_{ij})*** of the different query types in the mix
- Use a linear regression model

$$A_{ij} = \beta_{0\,j} + \sum_{k=1}^{T} \beta_{jk} N_{ik}$$

- Fit β's (regression coefficients) to runs of sample query mixes
  - ***Statistical modeling / Experiment-driven modeling***

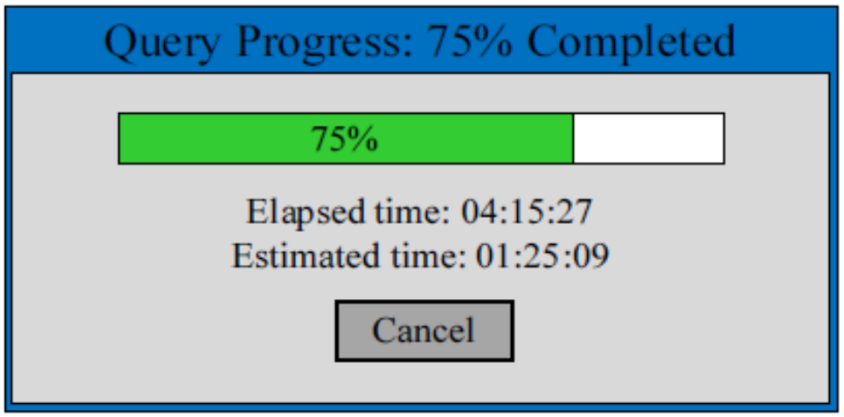
10GB TPC-H Database on DB2



- 10 instances of each of the 6 longest running TPC-H queries
- *p* is the skew in arrival order of queries (affects FCFS)
- MPL = 10
- Batch completion time is 107 minutes

Progress Monitoring

- Can be viewed as continuous on-line self-adjusting performance prediction
- Useful for ***workload monitoring*** and for making workload management decisions
- Starting point: ***query optimizer cost estimates***



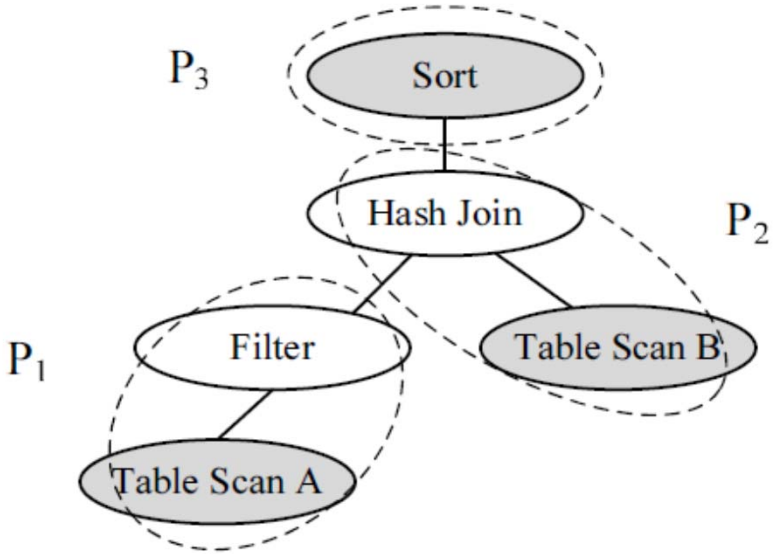
Solution Overview

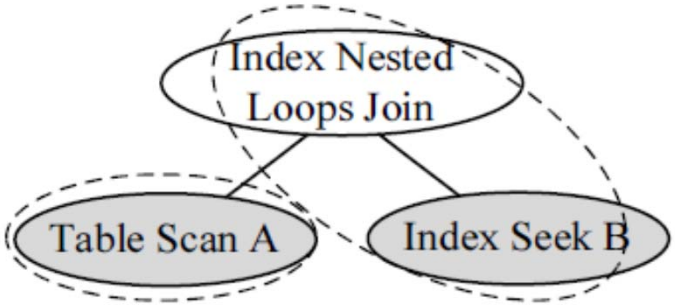
- First attempt at a solution :
  - Query optimizer estimates the number of tuples flowing through each operator in a plan.
  - Progress of a query =
    Total number of tuples that have flowed through different operators /
    Total number of tuples that will flow through all operators
- Refining the solution:
  - Take blocking behavior into account by dividing plan into independent ***pipelines***
  - More sophisticated estimate of the speed of pipelines
  - Refine estimated remaining time based on actual progress

Speed-independent Pipelines

Jiexing Li, Rimma V. Nehme, Jeffrey Naughton. "GSLPI: a Cost-based Query Progress Indicator." *ICDE*, 2012.

- Pipelines delimited by blocking or semi-blocking operators
- Every pipeline has a set of ***driver nodes***
- Pipeline execution follows a partial order

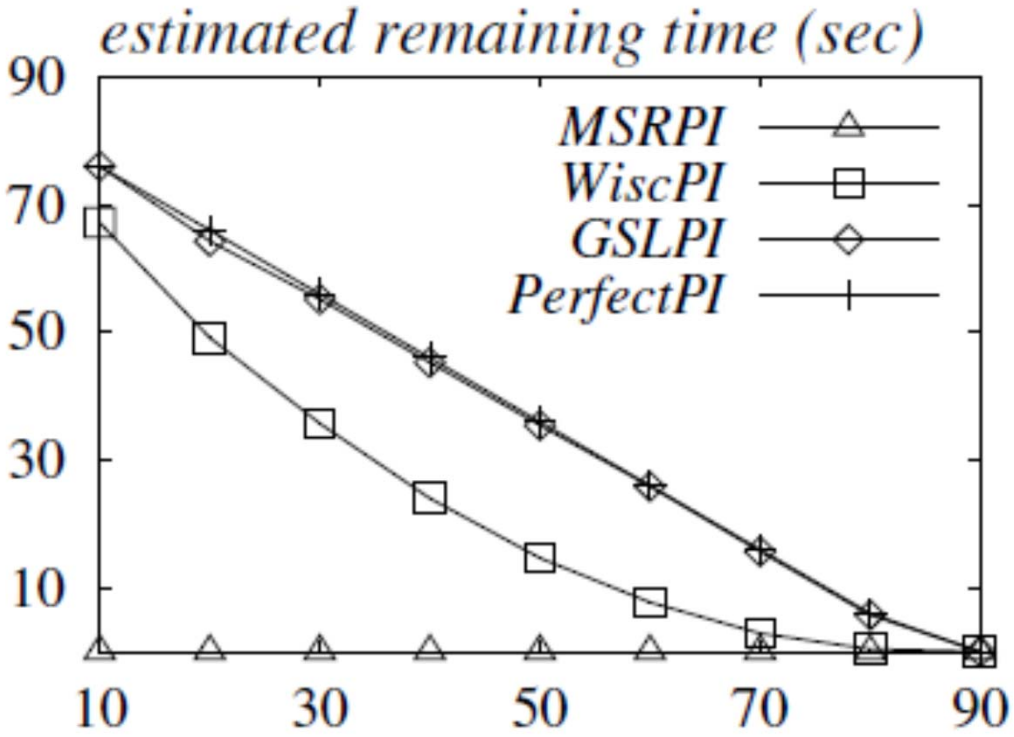




Estimating Progress

- Total time required by a pipeline
  - ***Wall-clock query cost:*** maximum amount of non-overlapping CPU and I/O
  - Based on query optimizer estimates
  - "Critical path"
- Pipeline speed: ***tuples processed per second*** for the last *T* seconds
  - Used to estimate remaining time for a pipeline
- Estimates of cardinality, CPU cost, and I/O cost refined as the query executes

Accuracy of Estimation

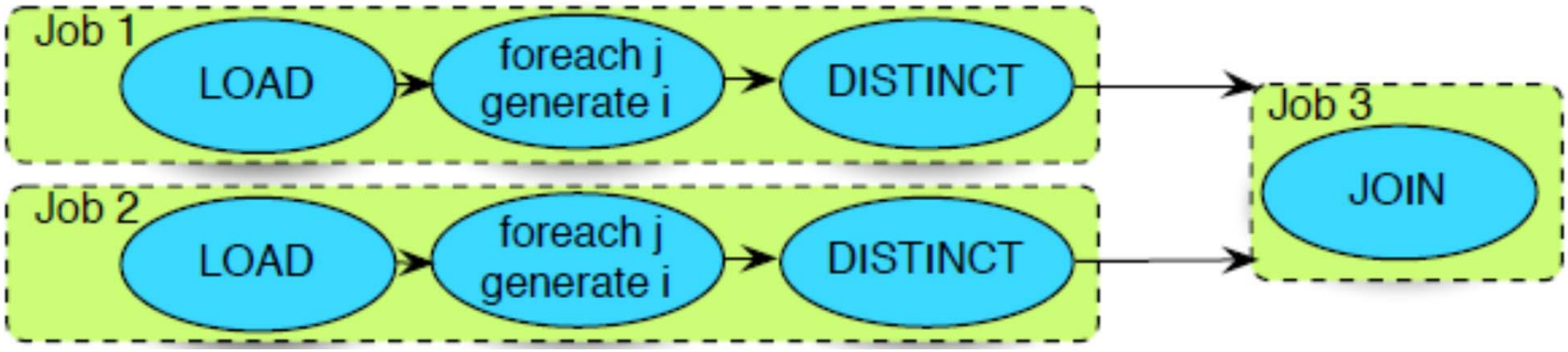


- Can use statistical models (i.e., machine learning) to choose the best progress indicator for a query
  Arnd Christian Konig, Bolin Ding, Surajit Chaudhuri, Vivek Narasayya. "A Statistical Approach Towards Robust Progress Estimation." *VLDB*, 2012.

Application to MapReduce

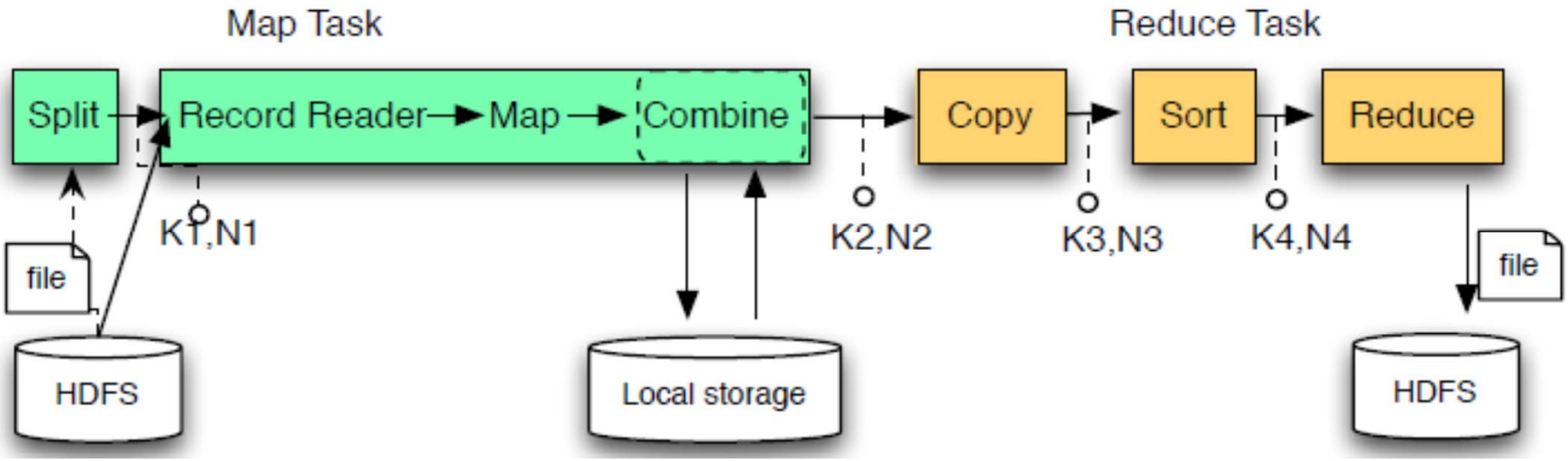
Kristi Morton, Magdalena Balazinska, Dan Grossman. "ParaTimer: A Progress Indicator for MapReduce DAGs." *SIGMOD*, 2010.

- Focuses on DAGs of MapReduce jobs produced from Pig Latin queries



MapReduce Pipelines

- Pipelines corresponding to the ***phases of execution*** of MapReduce jobs
- Assumes the ***existence of cardinality estimates*** for pipeline inputs
- Use observed ***per-tuple execution cost*** for estimating pipeline speed



Progress Estimation

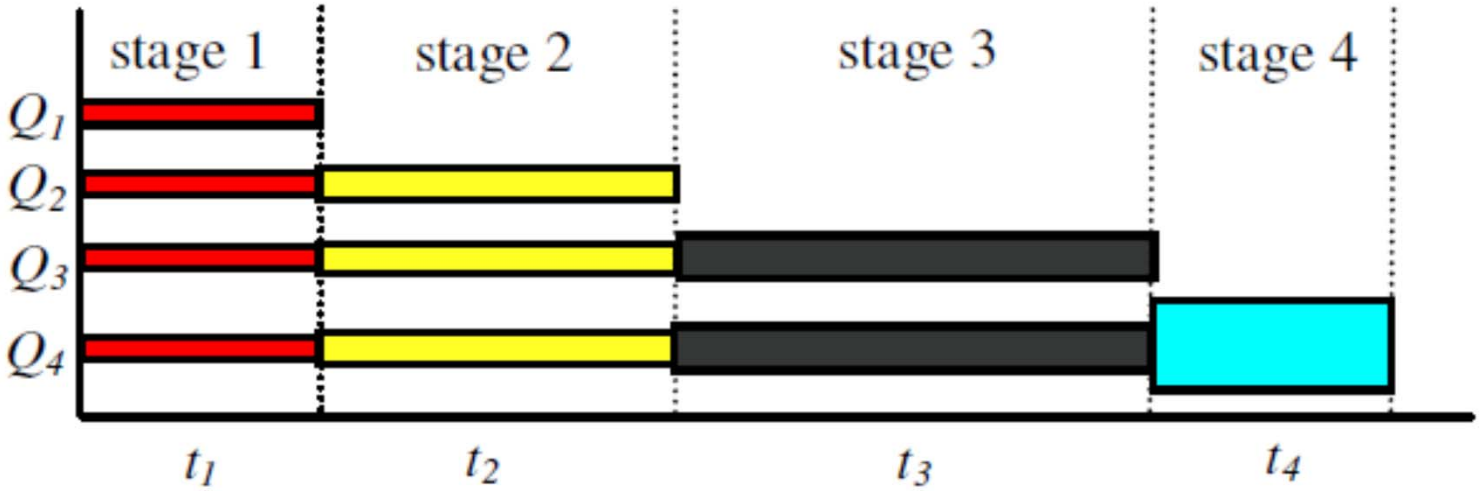
- Simulates the scheduling of Map and Reduce tasks to estimate progress

- Also provides an estimate of progress if ***failure*** were to happen during execution
  - Find the task whose failure would have the worst effect on progress, and report remaining time if this task fails ***(pessimistic)***
  - Adjust progress estimates if failures actually happen

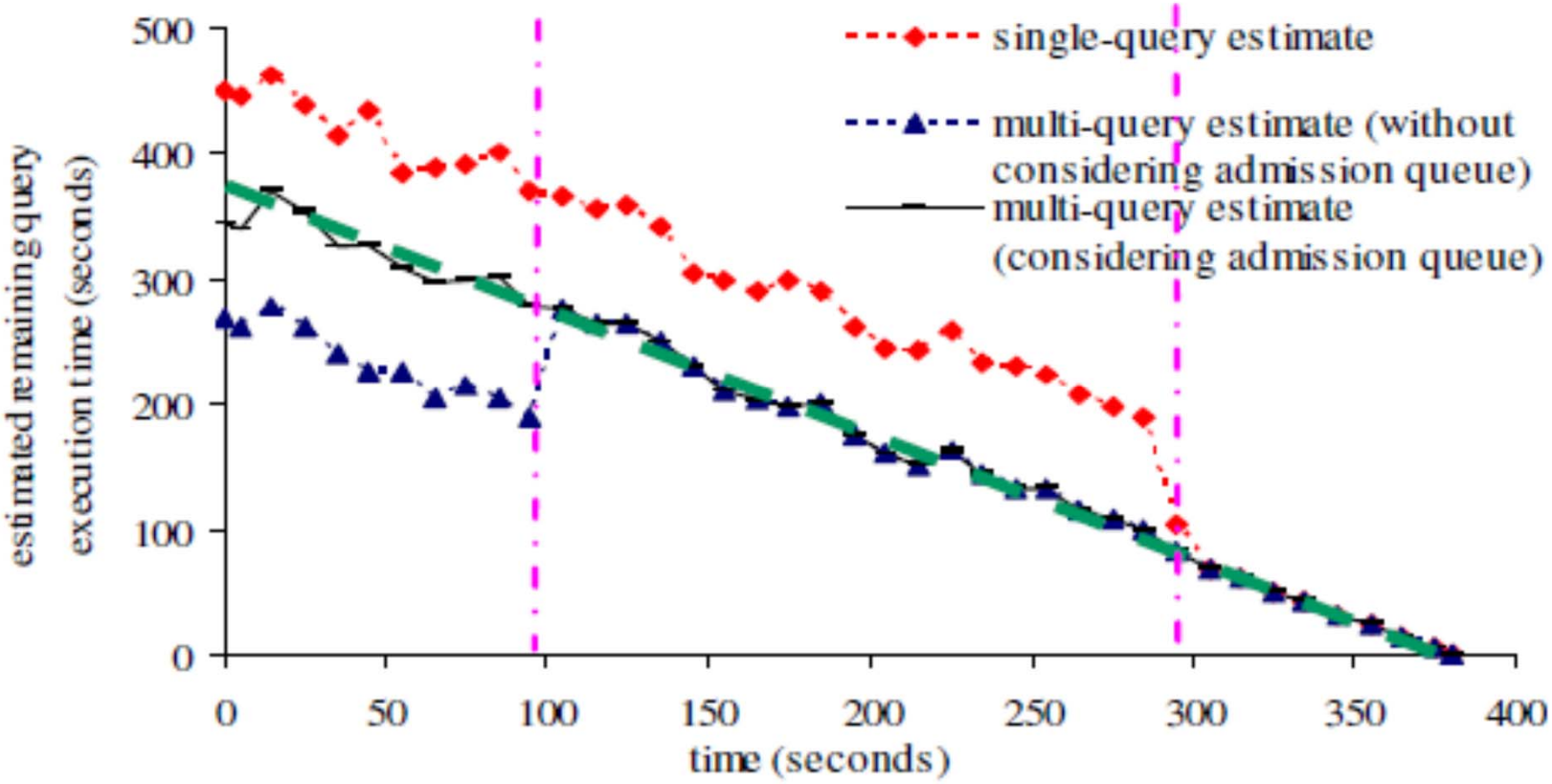
Progress of Interacting Queries

Gang Luo, Jeffrey F. Naughton, Philip S. Yu. "Multi-query SQL Progress Indicators." *EDBT*, 2006.

- Estimates the progress of multiple queries in the presence of ***query interactions***
  - The speed of a query is proportional to its ***weight***
  - Weight derived from query priority and available resources
  - When a query in the current ***query mix*** finishes, there are more resources available so the weights of remaining queries can be increased



Accuracy of Estimation



- Can observe ***query admission queue*** to extend visibility into the future

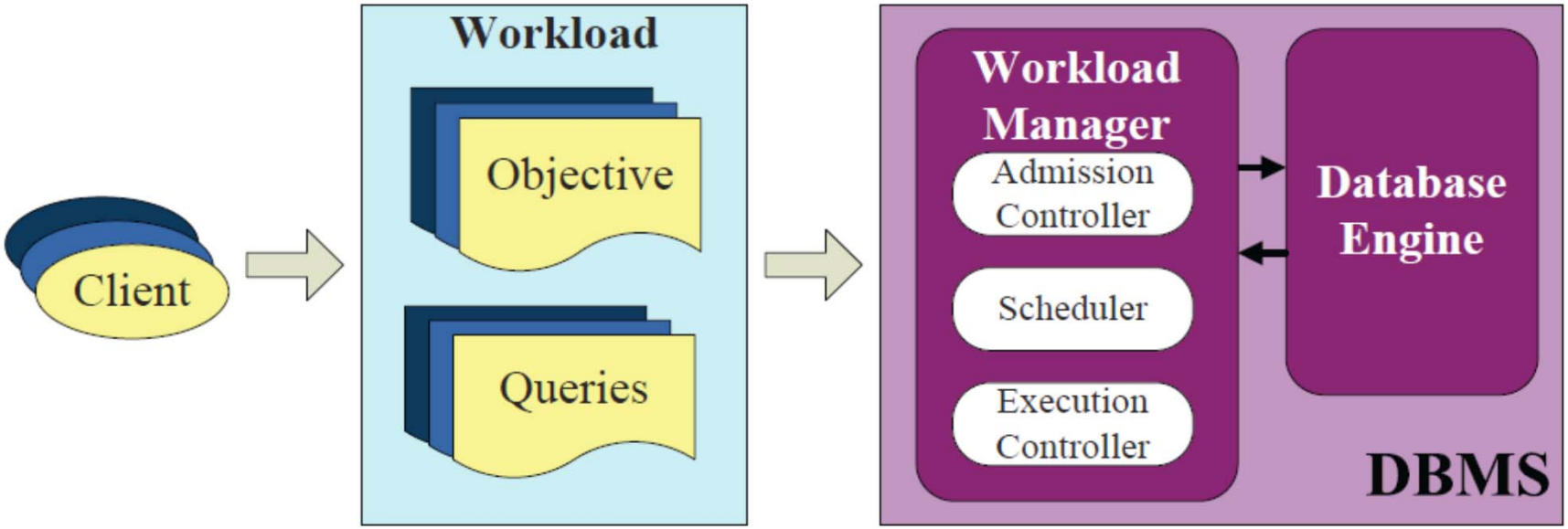
Relationship to WLM

- Can use the multi-query progress indicator to answer workload management questions such as
  - Which queries to block in order to speed up the execution of an important query?
  - Which queries to abort and which queries to wait for when we want to quiesce the system for maintenance?

Long-Running Queries

Stefan Krompass, Harumi Kuno, Janet L. Wiener, Kevin Wilkinson, Umeshwar Dayal, Alfons Kemper. "Managing Long-Running Queries." *EDBT*, 2009.

- A close look at the effectiveness of using admission control, scheduling, and execution control to manage long-running queries



Classification of Queries

	Query expected to be long	Query progress reasonable	Uses equal share of resources
expected-heavy	Yes	Yes	Equal share
expected-hog	Yes	Yes	> Equal share
surprise-heavy	No	Yes	Equal share
surprise-hog	No	Yes	> Equal share
overload	No	No	Equal share
starving	No	No	< Equal share

- Estimated resource shares and execution time based on ***query optimizer cost estimates***

Workload Management Actions

- Admission control
  - Reject, hold, or warn if estimated cost > threshold
- Scheduling
  - Two FIFO queues, one for queries whose estimated cost < threshold, and one for all other queries
  - Schedule from the queue of short-running queries first
- Execution control
  - Actions: Lower query priority, stop and return results so far, kill and return error, kill and resubmit, suspend and resume later
  - Supported by many commercial database systems
  - Take action if observed cost > threshold
  - Threshold can be absolute or relative to estimated cost (e.g., 1.2*estimated cost)

Surprise Queries

- Experiments based on simulation show that workload management actions achieve desired objectives except if there are ***surprise-heavy*** or ***surprise-hog*** queries

Need accurate prediction of execution time and resource consumption

Tutorial Outline

- Introduction
- Workload-level decisions in database systems
  - Physical design, Scheduling, Progress monitoring, Managing long running queries
- Performance prediction
- ***Break***
- Inter-workload interactions
- Outlook and open problems

Performance Prediction

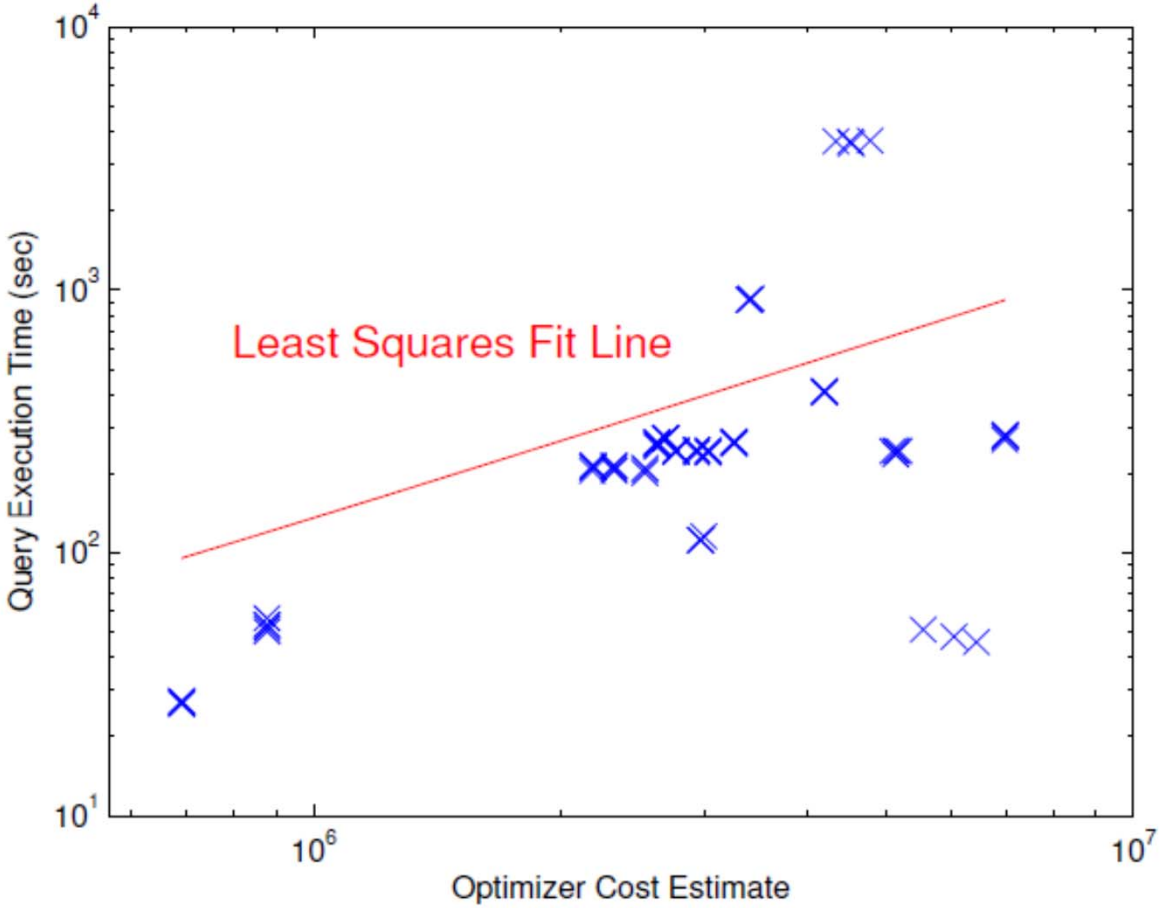
Performance Prediction

- Query optimizer estimates of query/operator cost and resource consumption are OK for choosing a good query execution plan
- These estimates ***do not correlate well*** with actual cost and resource consumption
  - But they can still be useful
- Build ***statistical / machine learning models*** for performance prediction
  - ***Which features?*** Can derive from query optimizer plan.
  - ***Which model?***
  - ***How to collect training data?***

Query Optimizer vs. Actual

Mert Akdere, Ugur Cetintemel, Matteo Riondato, Eli Upfal, Stanley B. Zdonik. “Learning-based Query Performance Modeling and Prediction.” *ICDE*, 2012.

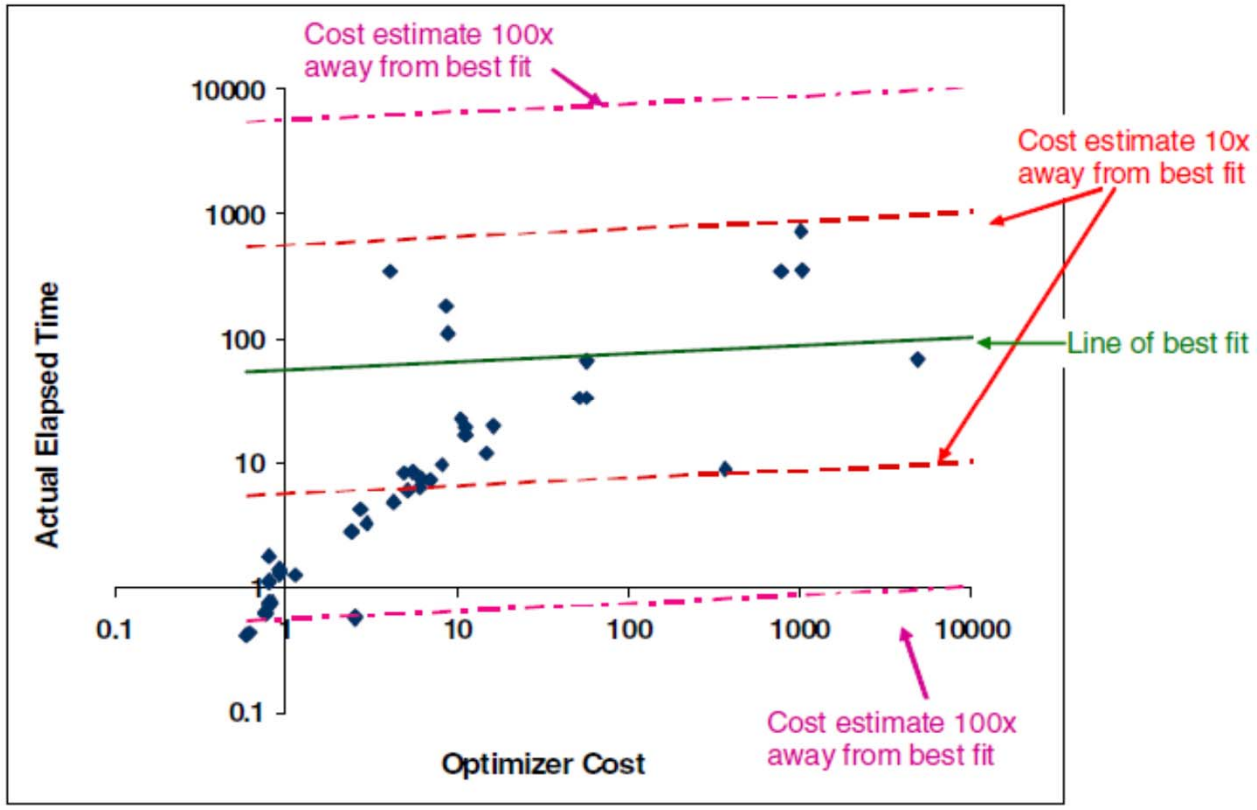
- 10GB TPC-H queries on PostgreSQL



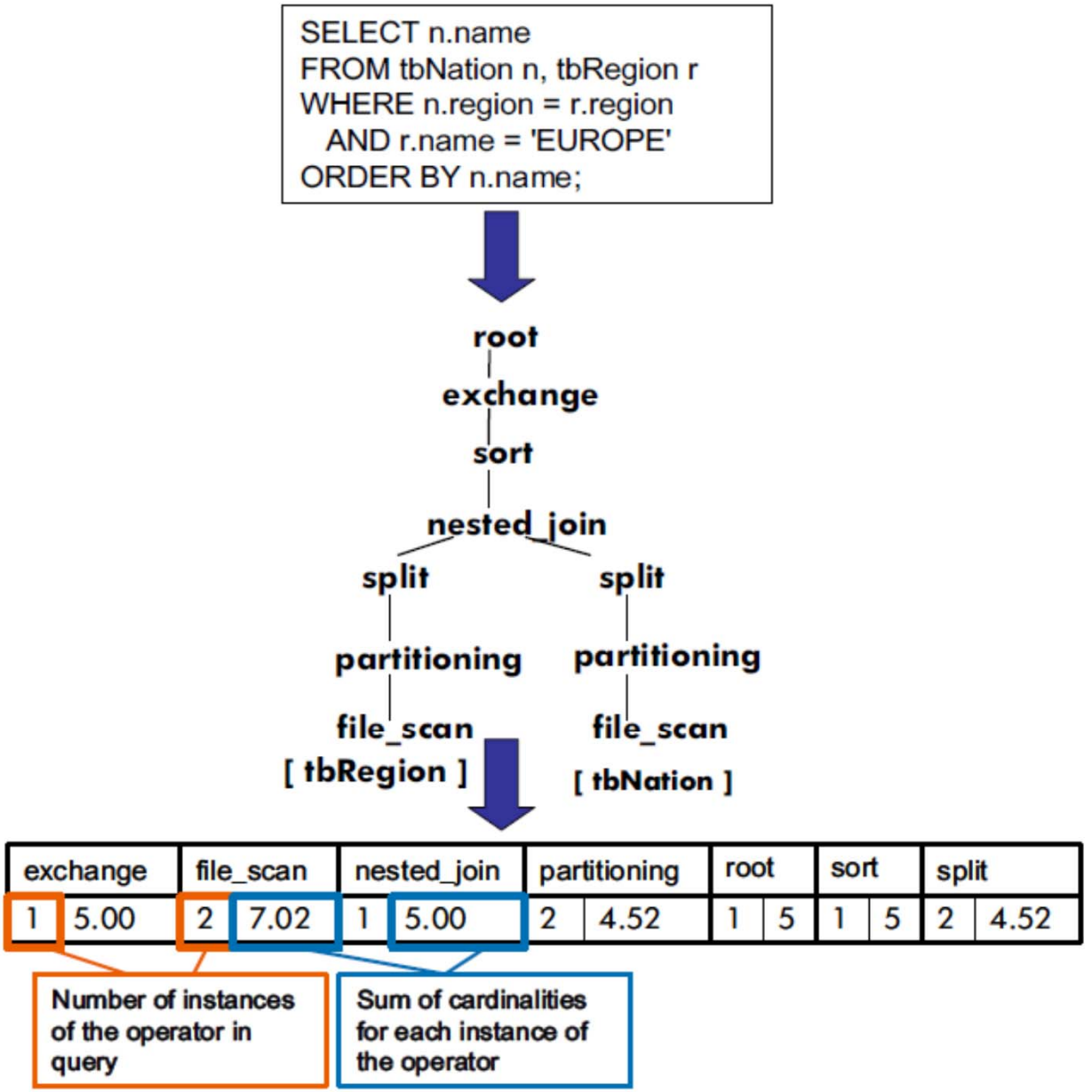
Prediction Using KCCA

Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox, Michael Jordan, David Patterson. "Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning." *ICDE*, 2009.

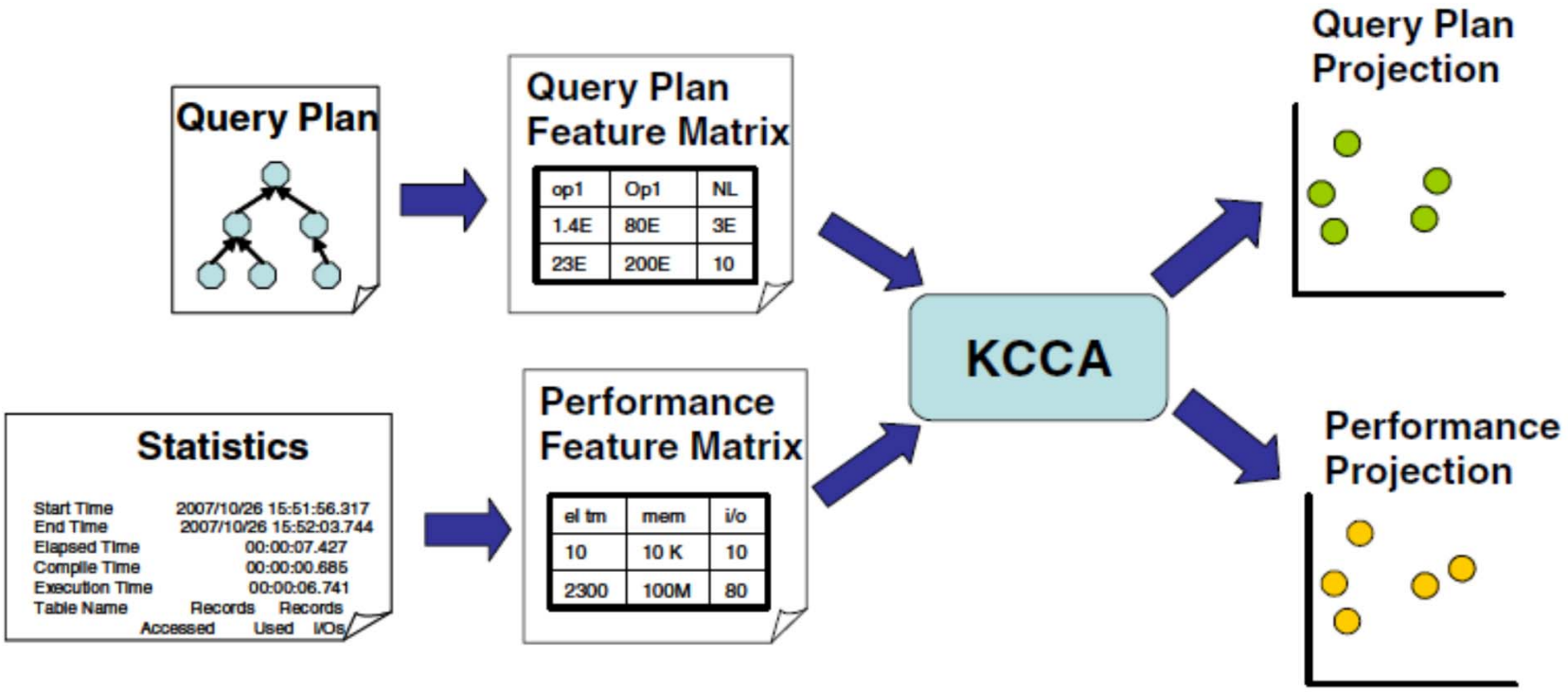
- Optimizer vs. actual: TPC-DS on Neoview



Aggregated Plan-level Features

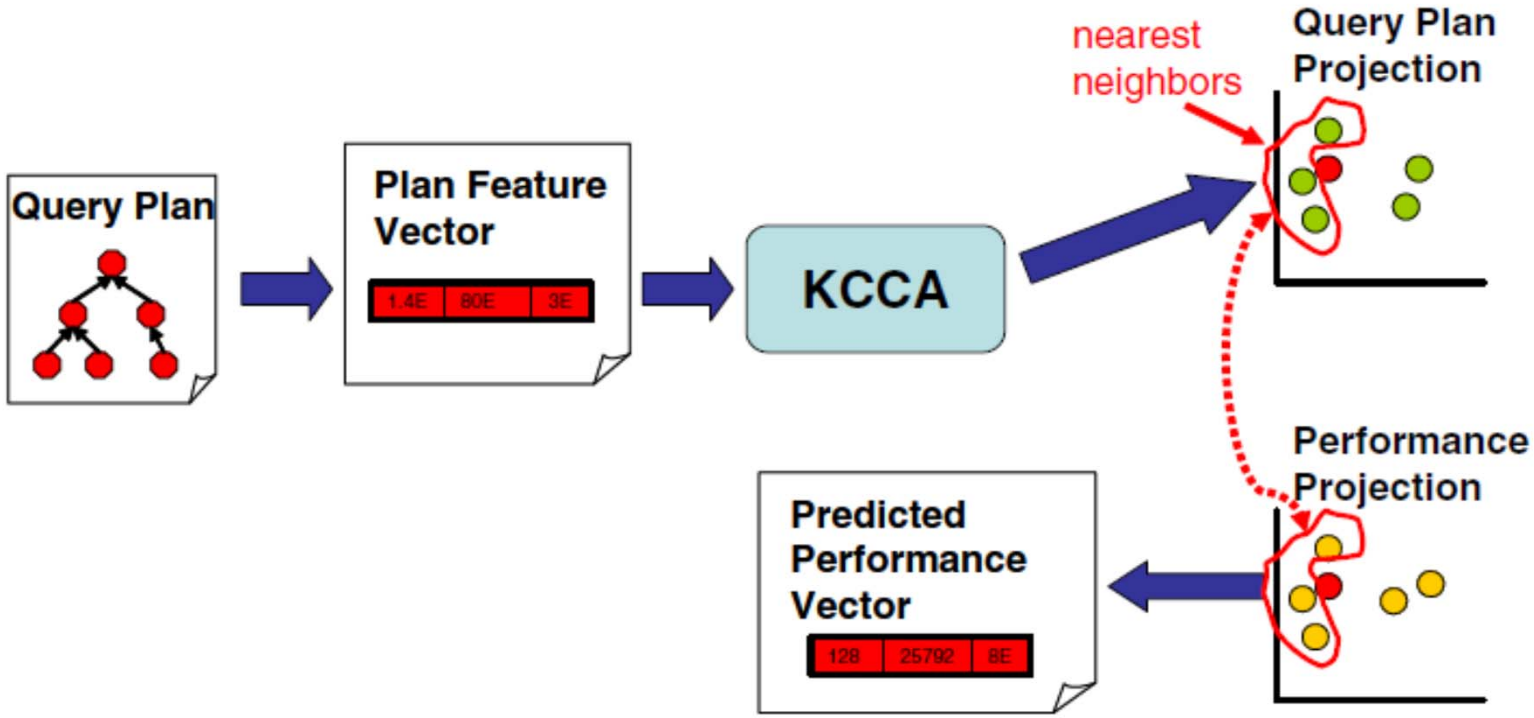


Training a KCCA Model



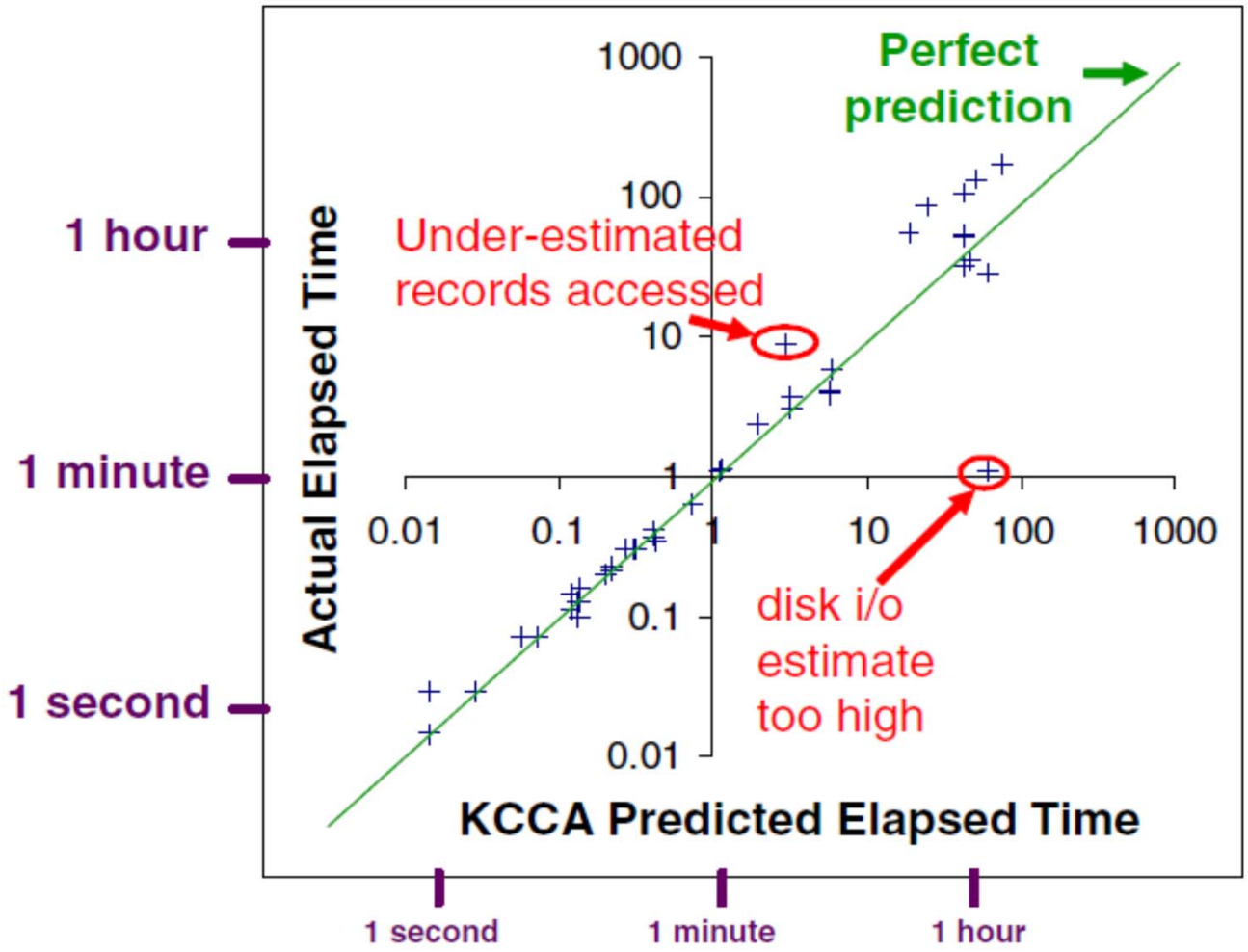
- Principal Component Analysis -> Canonical Correlation Analysis -> Kernel Canonical Correlation Analysis
- KCCA finds ***correlated pairs of clusters*** in the query vector space and performance vector space

Using the KCCA Model



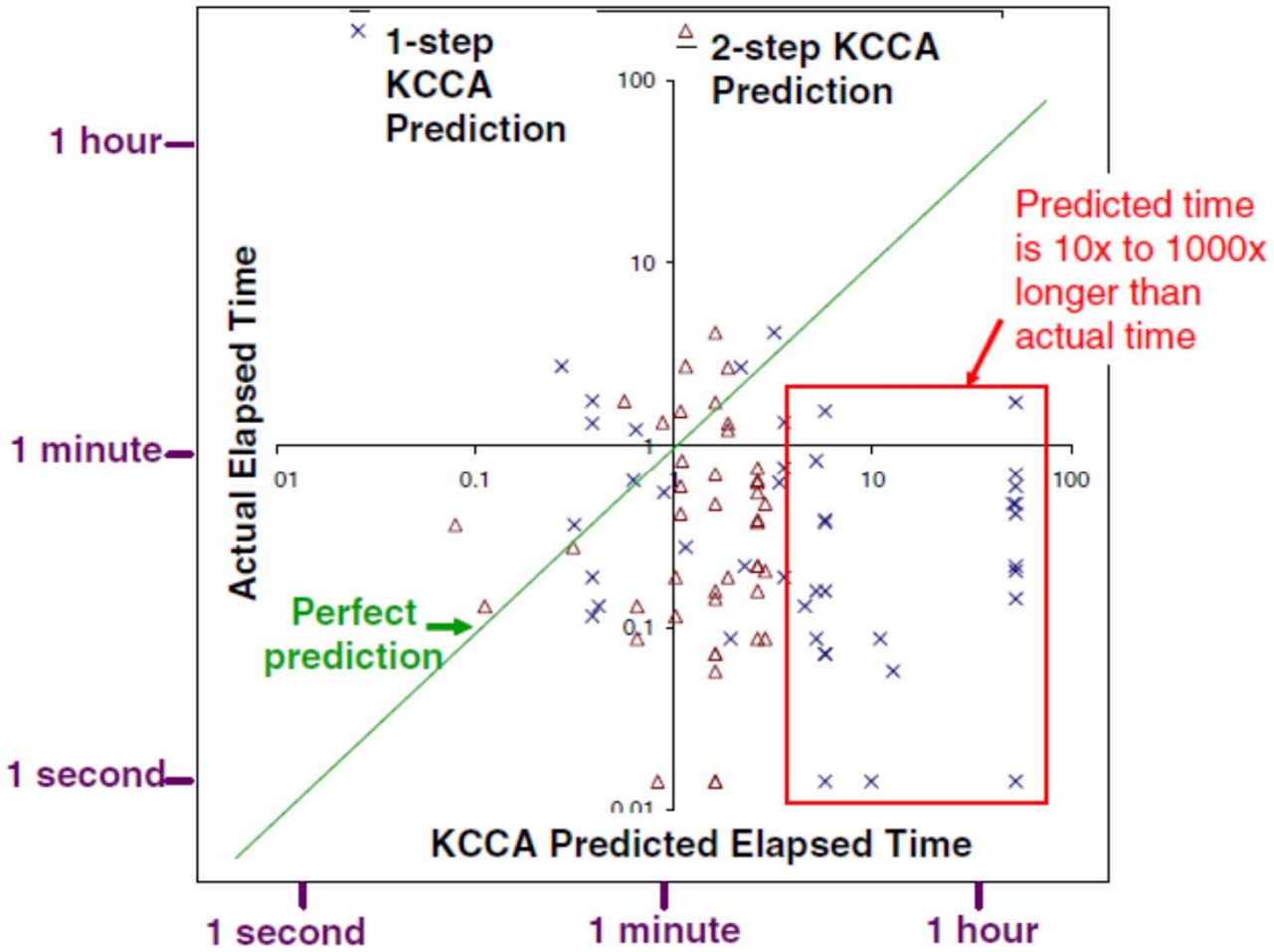
- Keep all projected query plan vectors and performance vectors
- Prediction based on ***nearest neighbor query***

Results: The Good News



- Can also predict records used, I/O, messages

Results: The Bad News

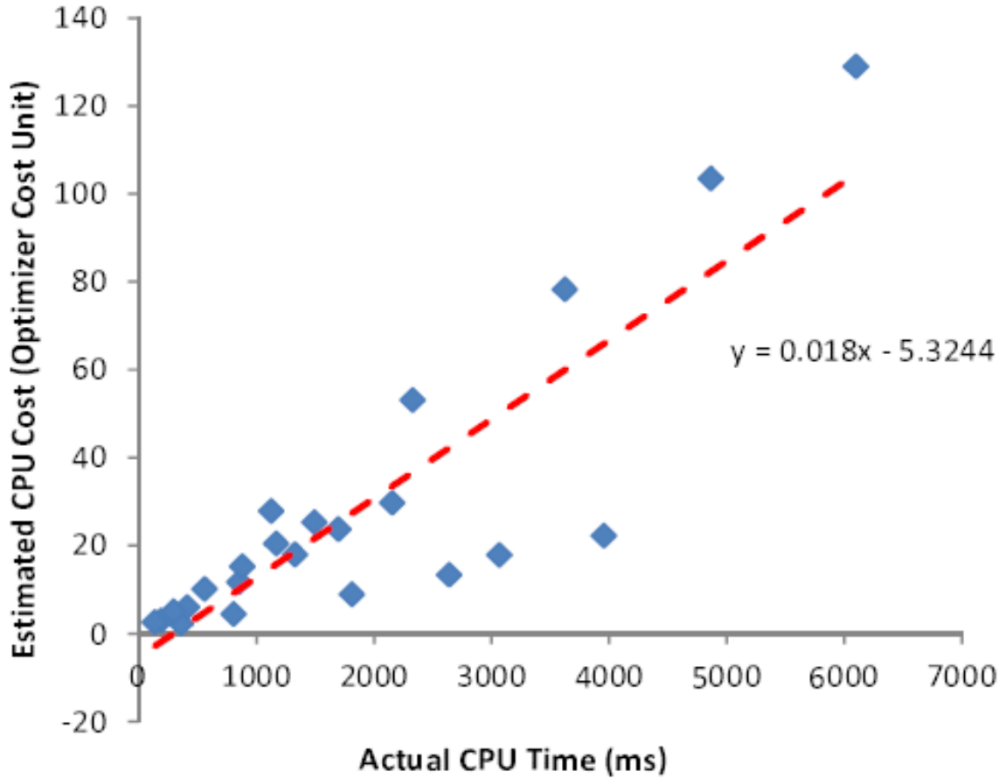


- Aggregate plan-level features cannot generalize to different schema and database

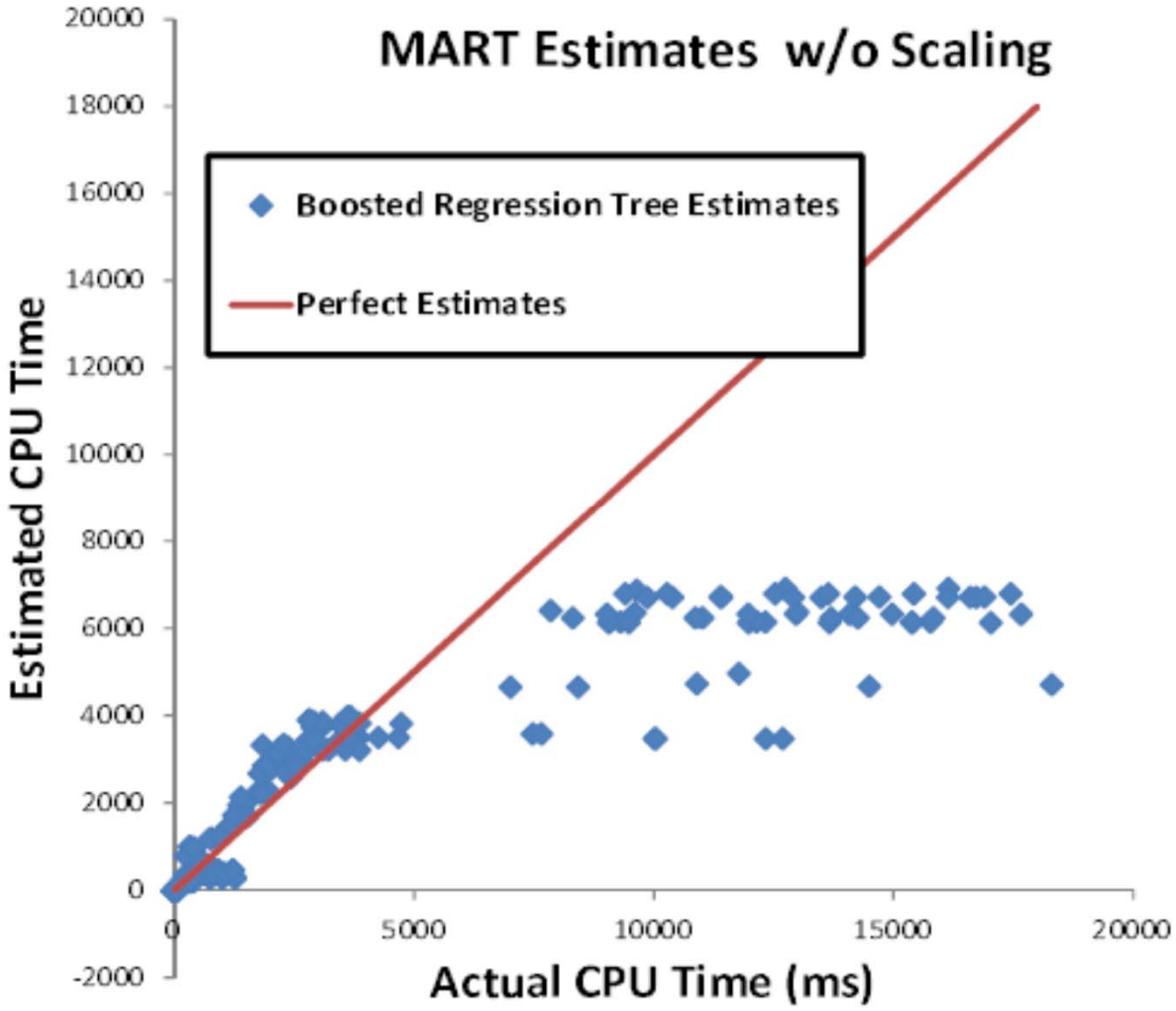
Operator-level Modeling

Jiexing Li, Arnd Christian Konig, Vivek Narasayya, Surajit Chaudhuri. “Robust Estimation of Resource Consumption for SQL Queries using Statistical Techniques.” *VLDB*, 2012.

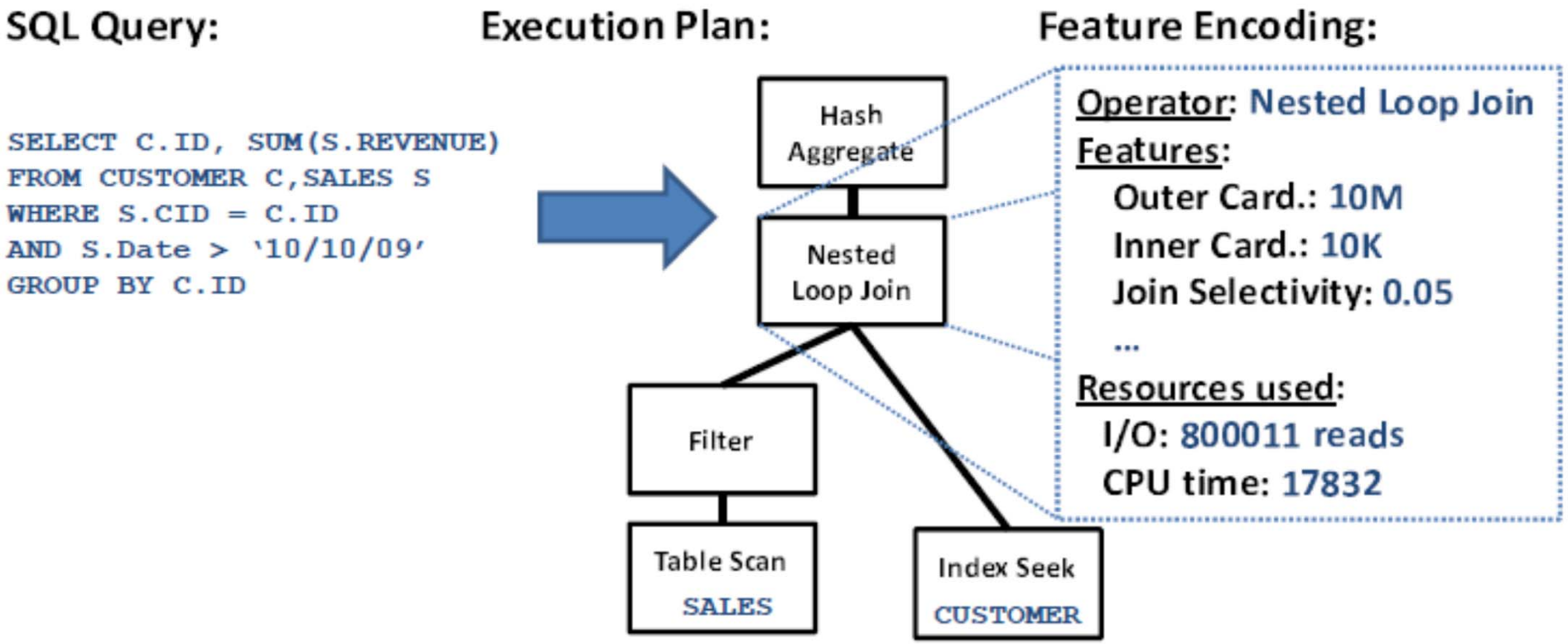
- Optimizer vs. actual CPU
  - With accurate cardinality estimates



Lack of Generalization



Operator-level Modeling



- One model for each type of query processing operator, based on features specific to that operator

Operator-specific Features

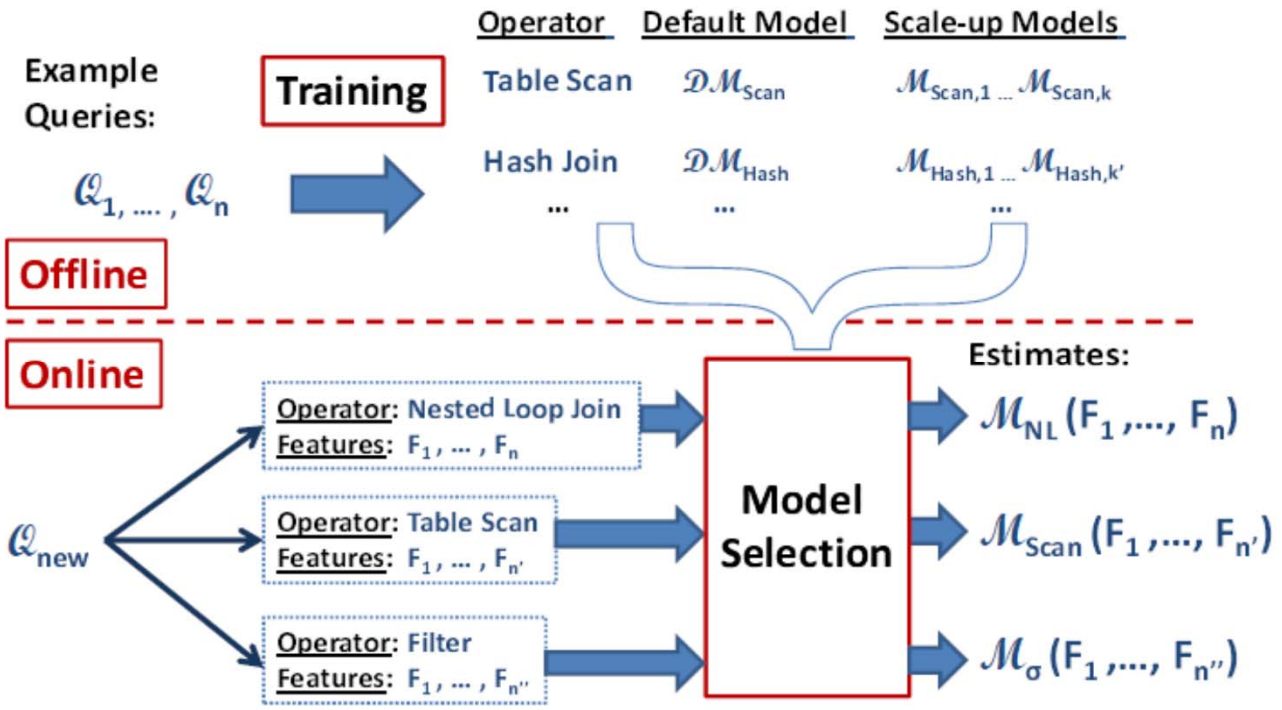
Name	Description	Notes
COUT	# of output tuples	
SOUTAVG	Avg. width of output tuples	
SOUTTOT	Total Number of bytes output	
CIN	# of input tuples	1 feature per child
SINAVG	Avg. width of input tuples	1 feature per child
SINTOT	Total number of bytes input	1 feature per child
OUTPUTUSAGE	Type of parent operator	Categorical Feature

Global Features (for all operator types)

Operator-specific Features

Name	Description	Operator
TSIZE	Size of input table in tuples	Seek/Scan
PAGES	Size of input table in pages	Seek/Scan
TCOLUMNS	Number of columns in a tuple	Seek/Scan
ESTIOCOST	Optimizer-estimated I/O cost	Seek/Scan
INDEXDEPTH	# Levels of Index in access path	Seek
HASHOPAVG	# Hashing operations per tuple	Hash Agg./Join
HASHOPTOT	HASHOPAVG $\times$ # Tuples	Hash Agg./Join
CHASHCOL	# columns involved in Hash	Hash Agg.
CINNERCOL	# columns involved in Join (Inner)	Joins
COUTERCOL	# columns involved in Join (Outer)	Joins
SSEEKTABLE	# Tuples in inner table	Nested Loop
MINCOMP	# Tuples $\times$ sort columns	Sort
CSORTCOL	# columns involved in Sort	Sort
SINSUM	Tot. bytes input in all children	Merge Join

Model Training



- Use ***regression tree models***
  - No need for dividing feature values into distinct ranges
  - No need for normalizing features (e.g, zero mean unit variance)
  - Different functions at different leaves, so can handle discontinuity (e.g., single-pass -> multi-pas sort)

Scaling for Outlier Features

- If feature *F* is much larger than all values seen in training, estimate resources consumed ***per unit F*** and scale using some feature- and operator-specific ***scaling function***
- Example: Normal CPU estimation

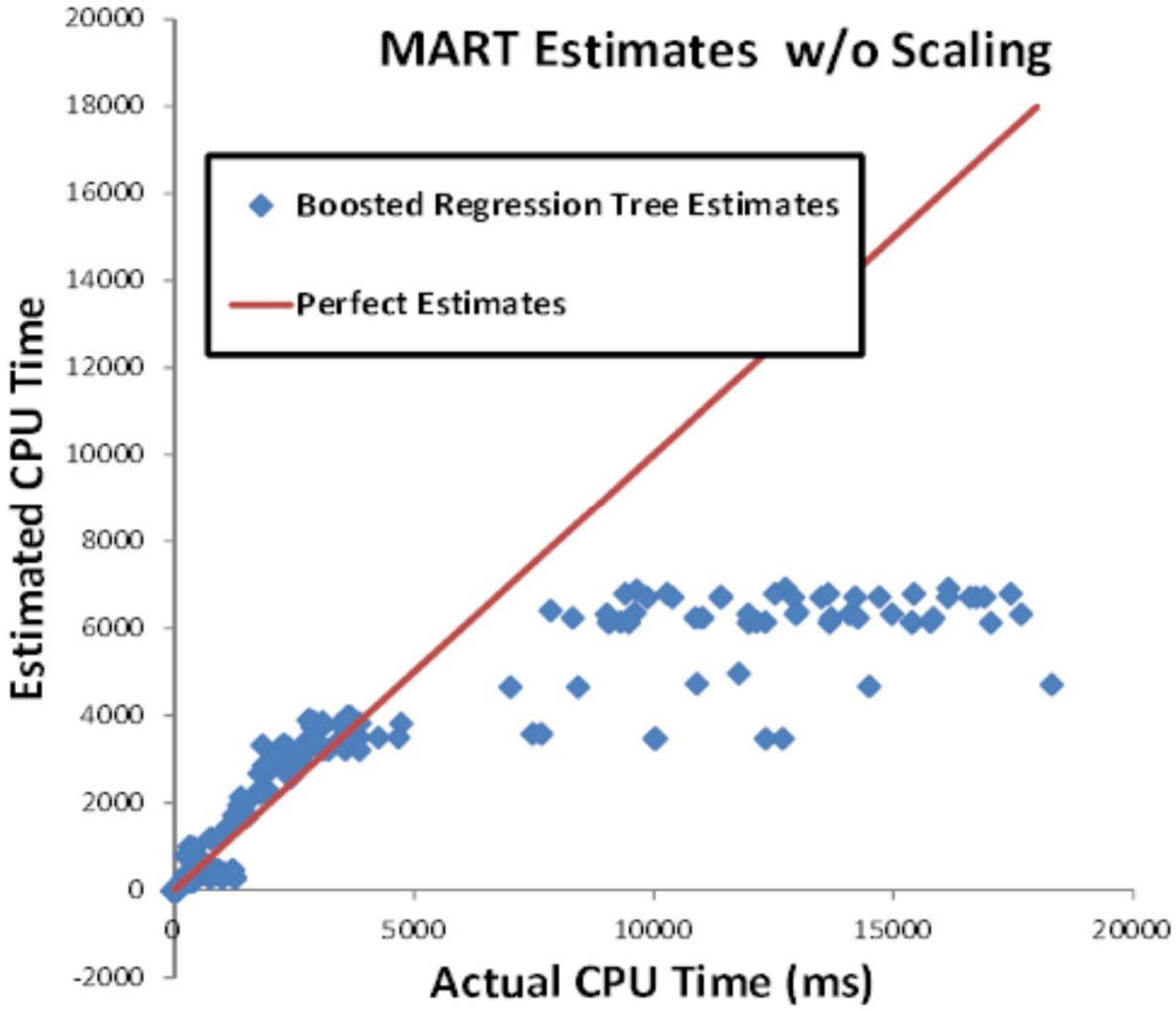
$$\mathcal{M}(\text{CIN}, \text{SINAVG}, \text{COUT}) \rightarrow CPU$$

- If CIN too large

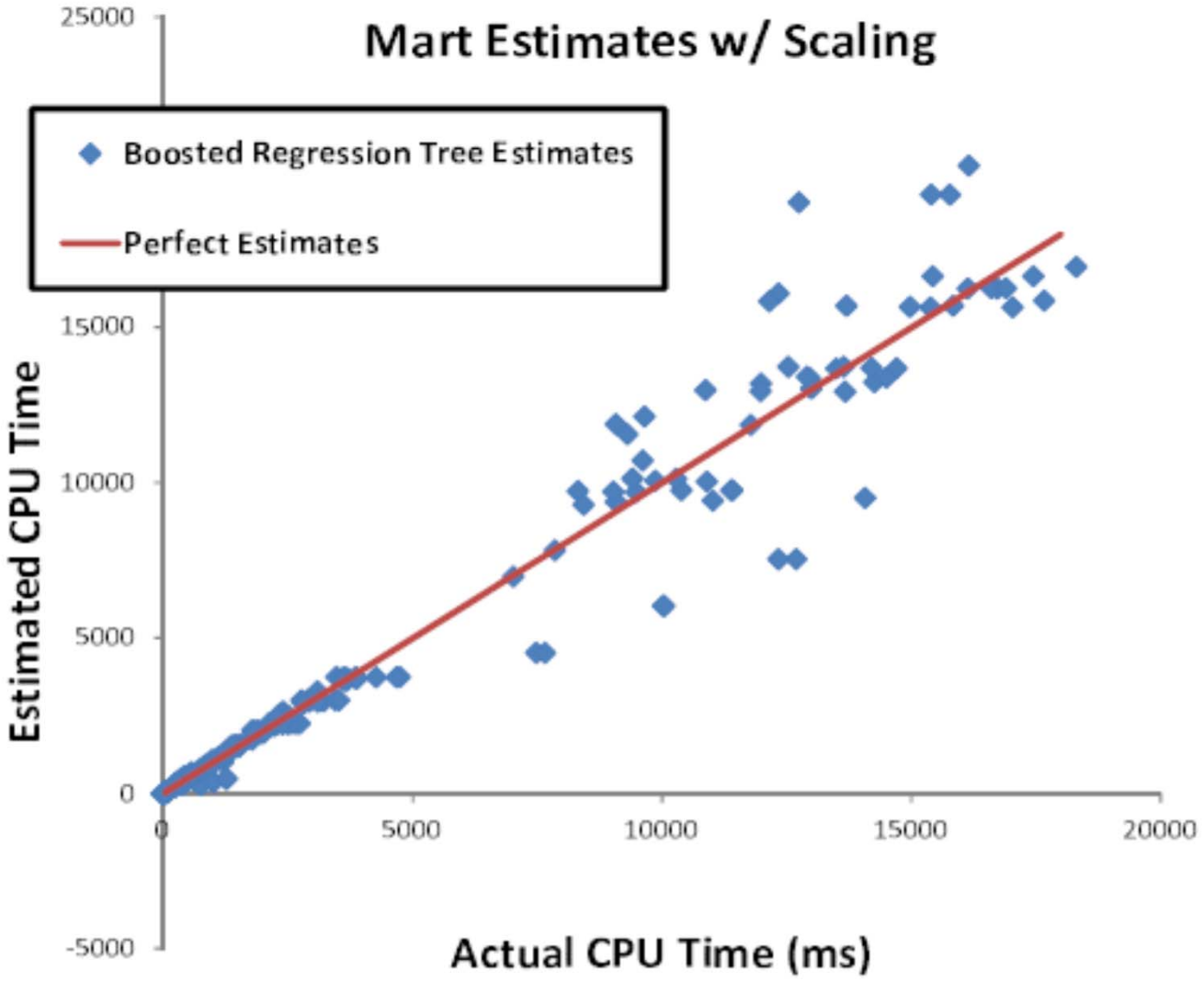
$$\mathcal{M}'(\text{SINAVG}, \text{COUT}) \rightarrow \frac{CPU}{\text{CIN}}$$

$$\begin{aligned} CPU &= SCALE_{\text{CPU},\text{CIN}}(\mathcal{M}'(\text{SINAVG}, \text{COUT})) \\ &= \underbrace{\text{CIN} \times}_{\text{Scaling function}} \underbrace{\mathcal{M}'(\text{SINAVG}, \text{COUT})}_{\text{Scaled Model}}. \end{aligned}$$

Accuracy Without Scaling



Accuracy With Scaling



Revisiting Optimizer Estimates

Wentao Wu, Yun Chi, Shenghuo Zhu, Junichi Tatemura, Hakan Hacigümüs, Jeffrey F. Naughton. "Predicting Query Execution Time: Are Optimizer Cost Models Really Unusable?" *ICDE,* 2013

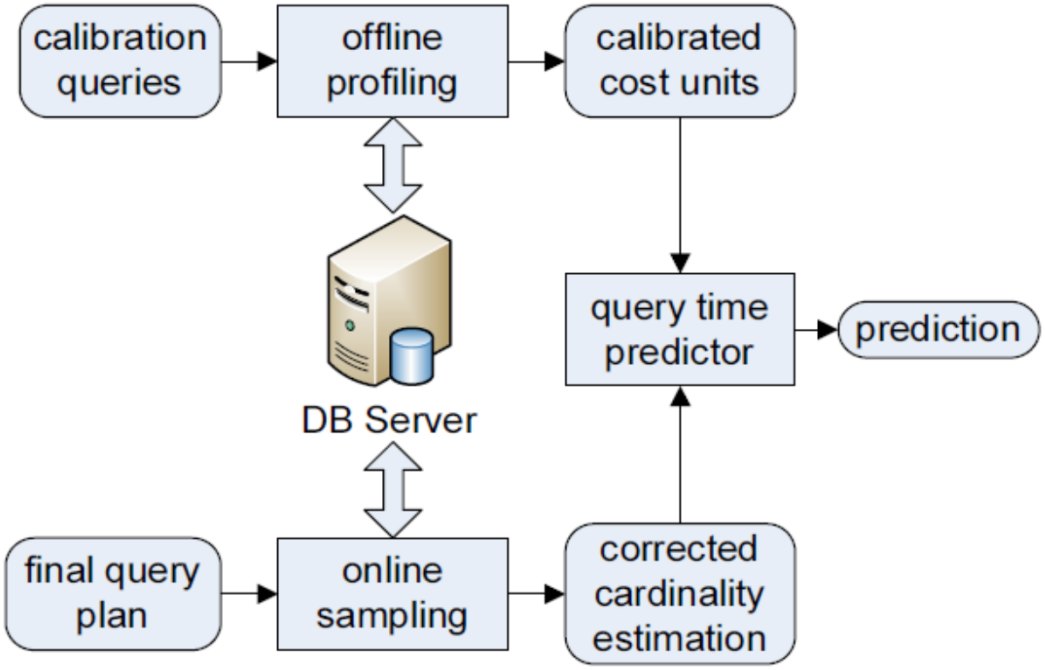
- With proper ***calibration*** of the query optimizer cost model, plus ***improved cardinality estimates***, the query optimizer cost model can be a good predictor of query execution time
- Example: PostgreSQL query optimizer cost equation

$$C_O = \mathbf{n}^\intercal \mathbf{c} = n_s \cdot c_s + n_r \cdot c_r + n_t \cdot c_t + n_i \cdot c_i + n_o \cdot c_o$$

  where *n*'s are pages accessed and *c*'s are calibration constants
- Good *n*'s and *c*'s will result in a good predictor

Calibration Plus Sampling

- A fixed set of queries to calibrate the cost model ***offline*** for the given hardware and software configuration
- Sampling to refine the cardinality estimates of the ***one plan*** chosen by the optimizer



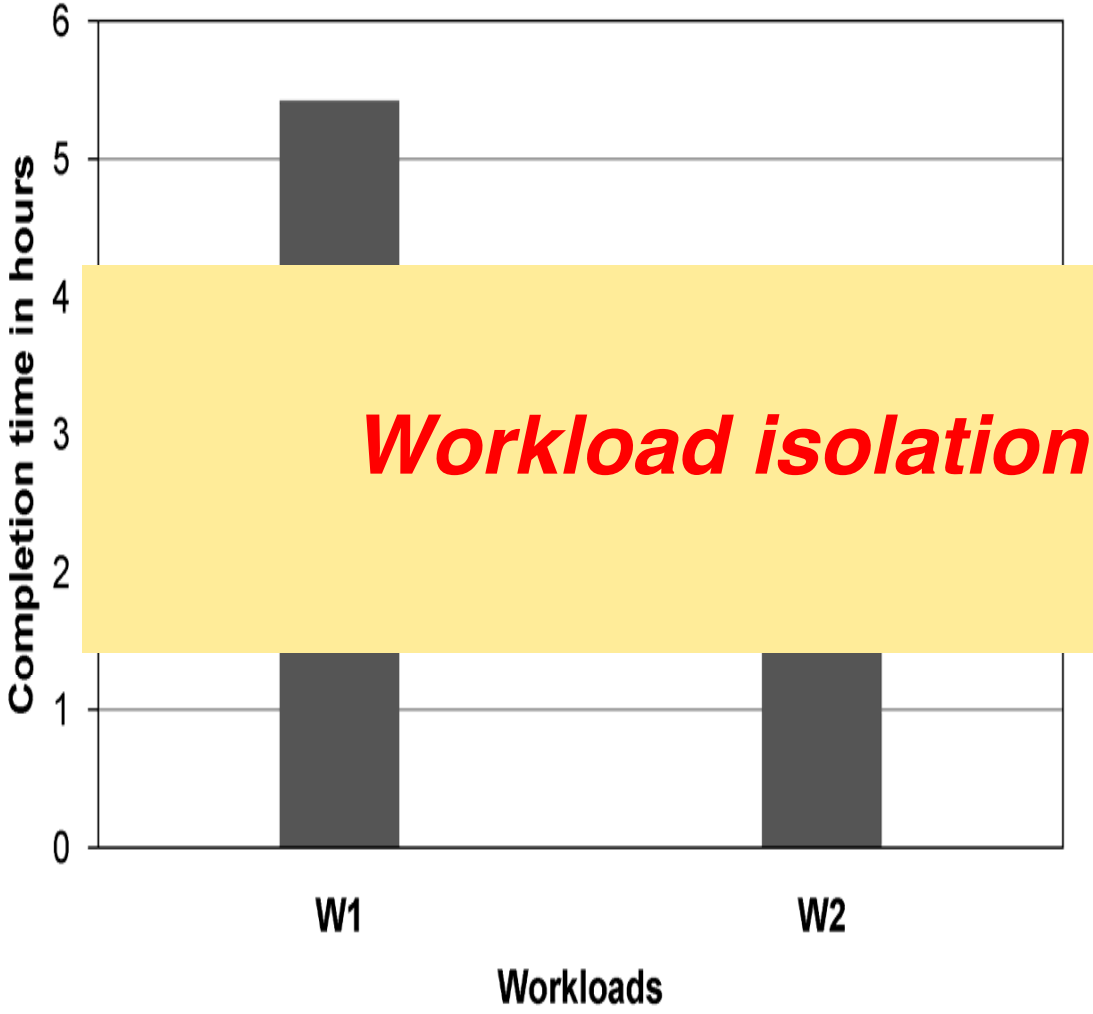
Modeling Query Interactions

Mumtaz Ahmad, Songyun Duan, Ashraf Aboulnaga, Shivnath Babu. "Predicting Completion Times of Batch Query Workloads Using Interaction-aware Models and Simulation." *EDBT*, 2011.

- A database workload consists of a sequence of mixes of ***interacting queries***
- Interactions can be significant, so their effects should be modeled
- Features = ***query types*** (no query plan features from the optimizer)
- A ***mix*** $m = <N_1, N_2, \ldots, N_T>$, where N_i is the number of queries of type i in the mix

Impact of Query Interactions

5.4 hours

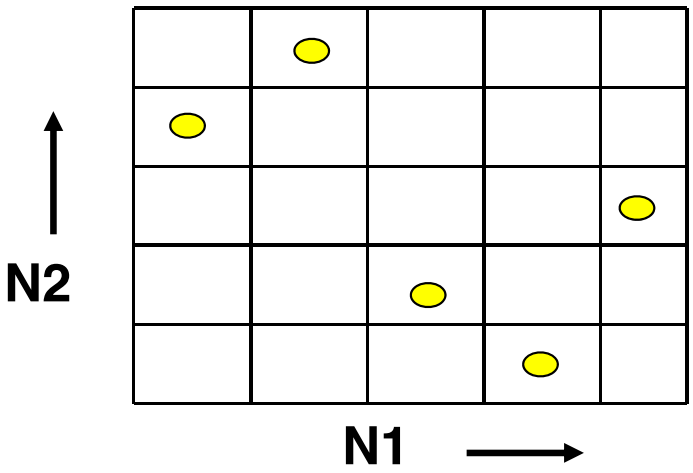


- Two workloads on a scale factor 10 TPC-H database on DB2

Workload isolation is important!

Sampling Query Mixes

- Query interactions complicate collecting a representative yet small set of training data
  - Number of possible query mixes is exponential
  - How judiciously use the available ***"sampling budget"***
- Interaction-level aware ***Latin Hypercube Sampling***
  - Can be done incrementally



Mix	Q_1		Q_7		Q_9		Q_{18}	
	N_i	A_i	N_i	A_i	N_i	A_i	N_i	A_i
m1	1	75	2	67	5	29.6	2	190
m2	4	92.3	0	0	0	0	1	53.5

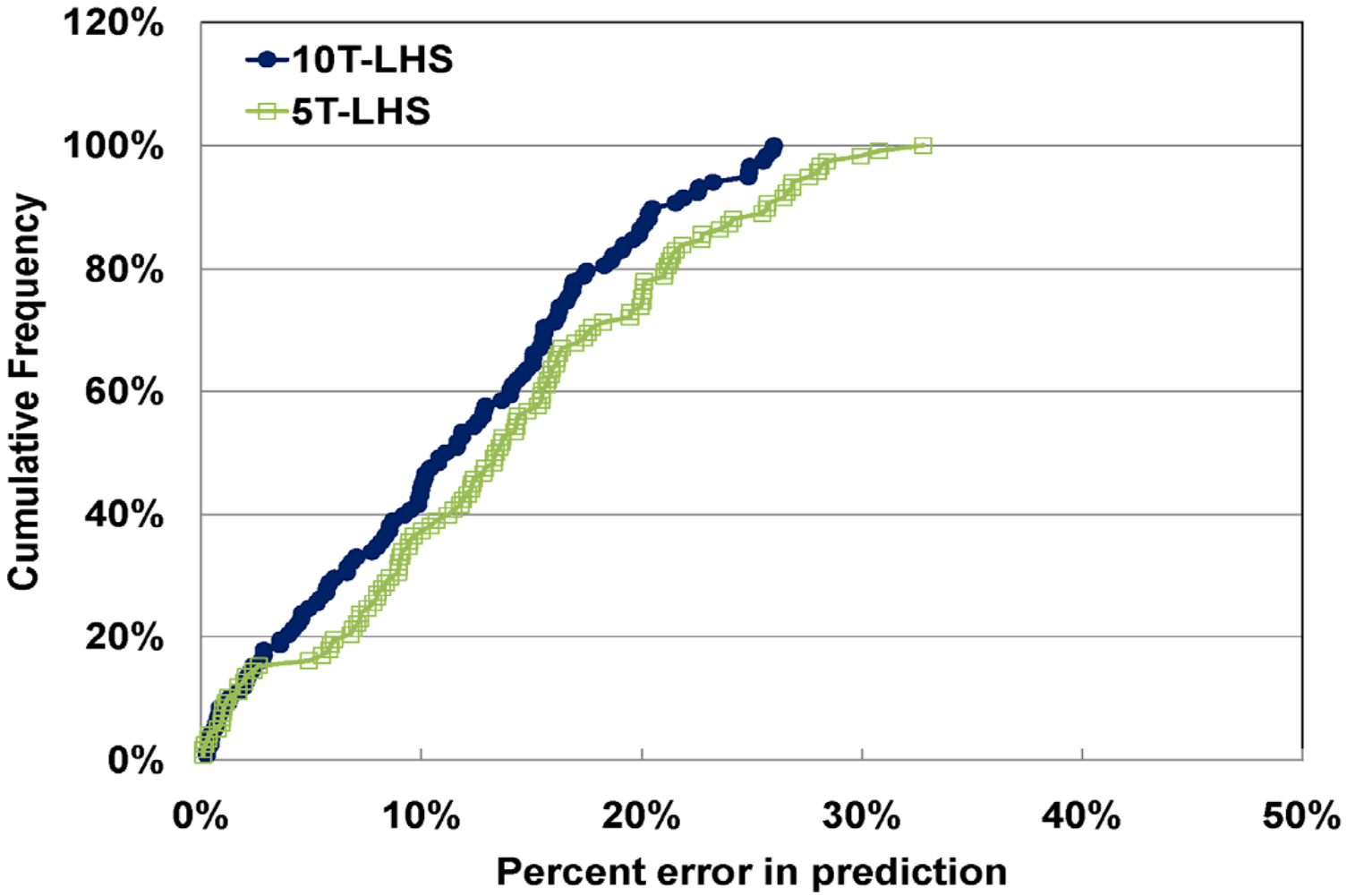
Interaction levels: m1=4, m2=2

Modeling and Prediction

- Training data used to build ***Gaussian Process Models*** for different query type
  - Model: CompletionTime (QueryType) = f(QueryMix)

- Models used in a simulation of workload execution to predict workload completion time

Prediction Accuracy



- Accuracy on 120 different TPC-H workloads on DB2

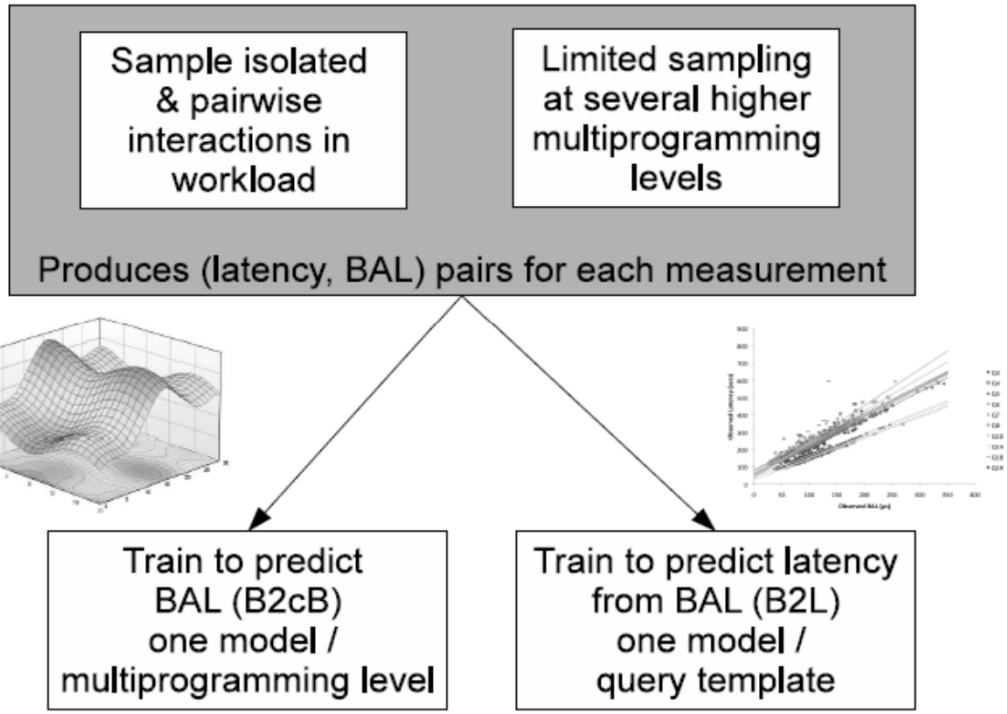
Buffer Access Latency

Jennie Duggan, Ugur Cetintemel, Olga Papaemmanouil, Eli Upfal. "Performance Prediction for Concurrent Database Workloads." *SIGMOD*, 2011.

- Also aims to model the effects of query interactions
- Feature used: ***Buffer Access Latency (BAL)***
  - The average time for a logical I/O for a query type
- Focus on sampling and modeling ***pairwise interactions*** since they capture most of the effects of interaction

Solution Overview

Training Phase



Prediction Phase

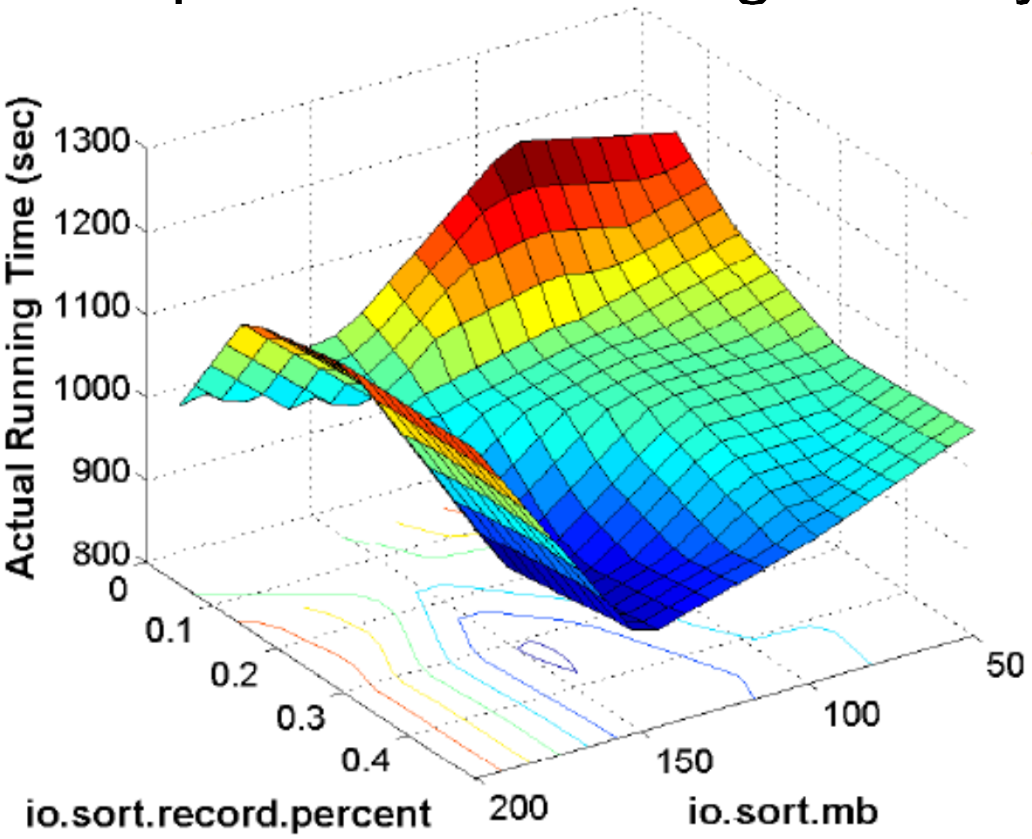
$$BAL_q = \alpha T_q + \beta \sum_{i=1}^{n} T_{c_i}$$

$$+\gamma_1 \sum_{i=1}^{n} \Delta T_{q/c_i} + \gamma_2 \sum_{i=1}^{n} \sum_{j=1}^{n,j\neq i} \Delta T_{c_i/c_j} \qquad L_q = O_q + P_q BAL_q$$

Prediction for MapReduce

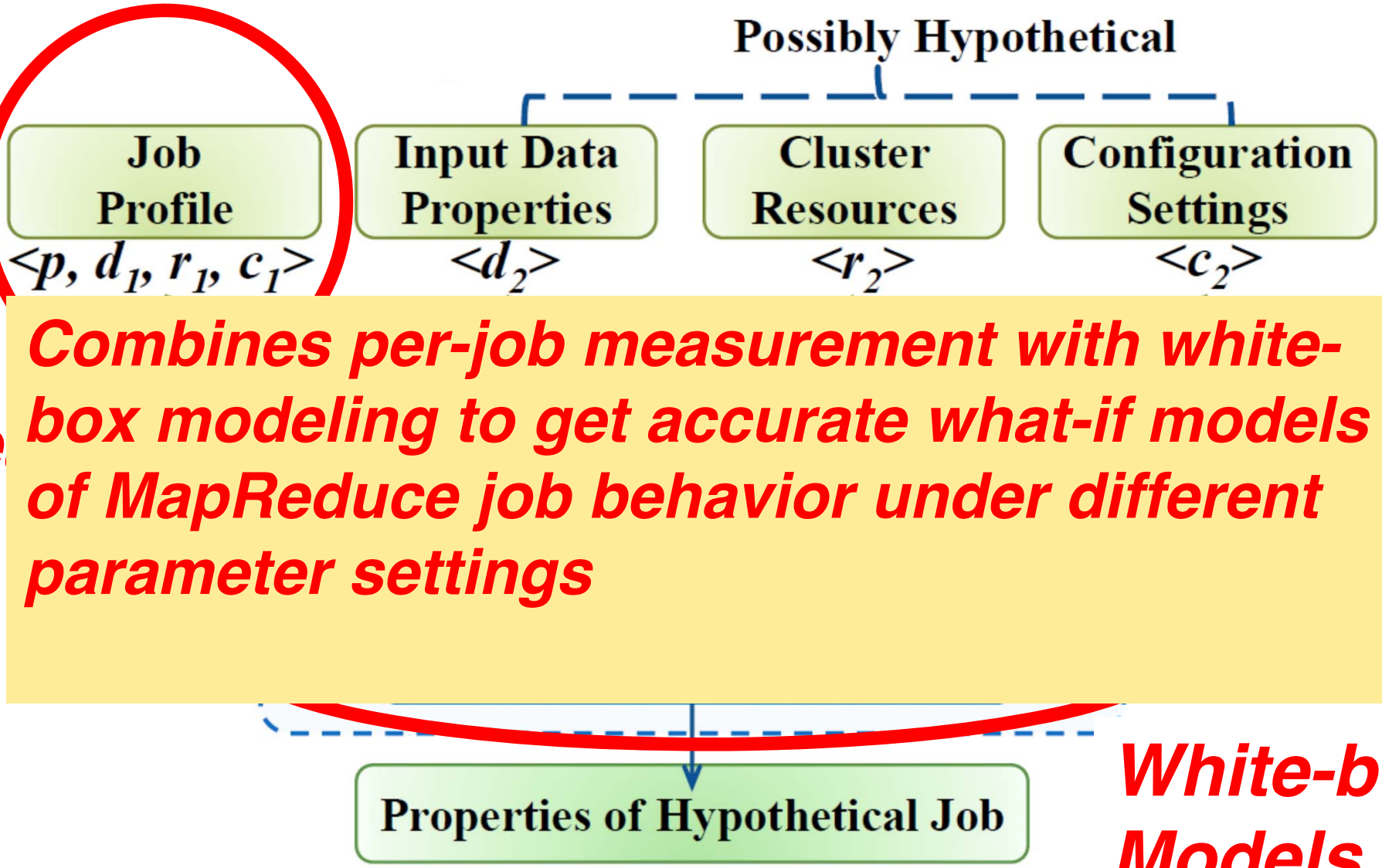
Herodotos Herodotou, Shivnath Babu. “Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs.” *VLDB*, 2011.

- Focus: Tuning MapReduce job parameters in Hadoop
- 190+ parameters that significantly affect performance



2-dim projection
of 13-dim surface

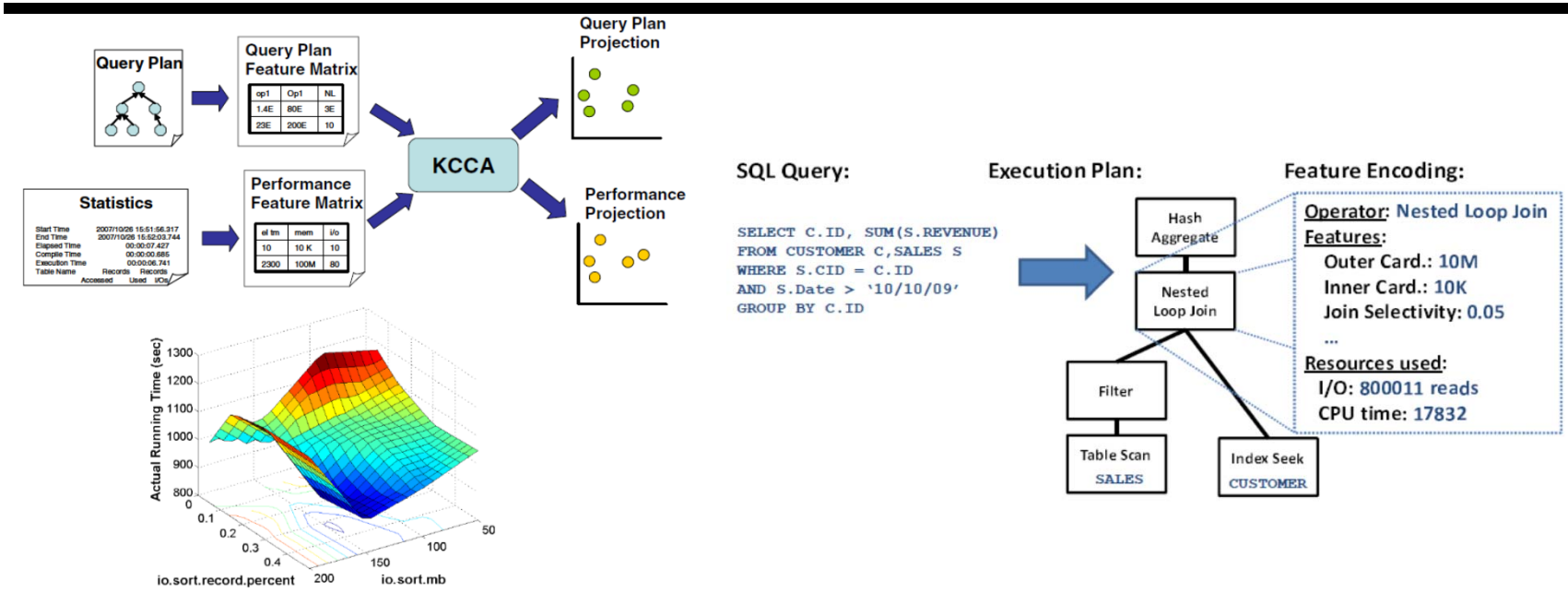
Starfish What-if Engine



Combines per-job measurement with white-box modeling to get accurate what-if models of MapReduce job behavior under different parameter settings

White-box Models

Recap



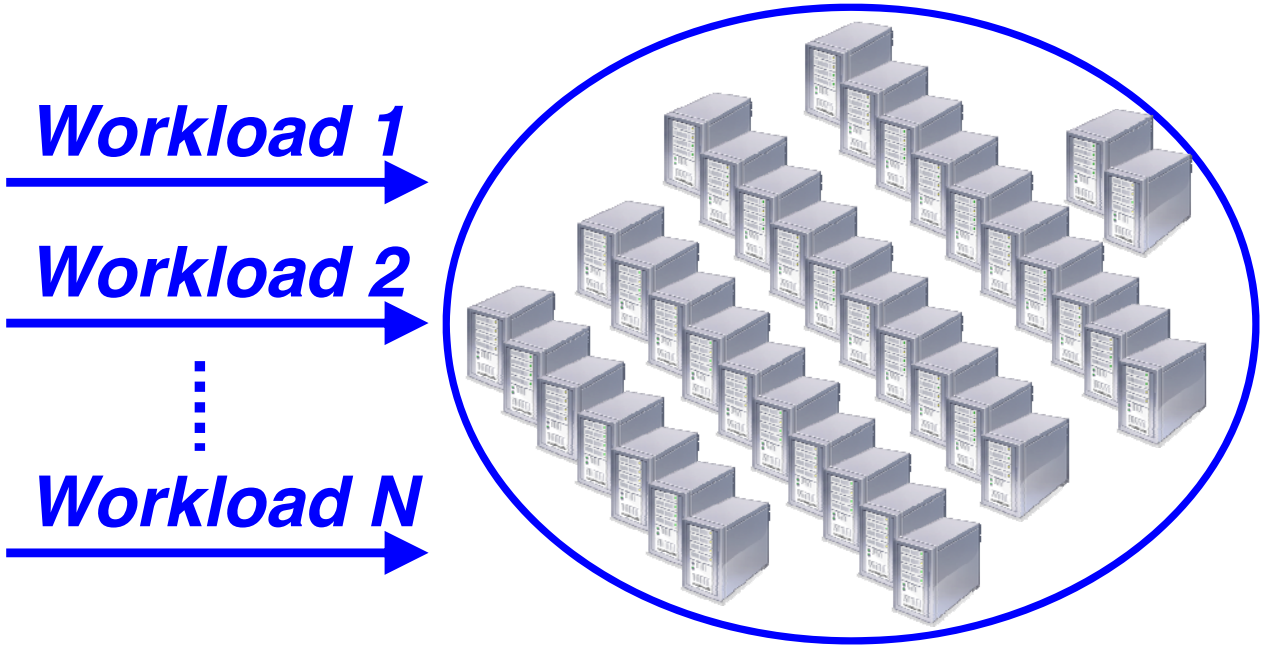
- ***Statistical / machine learning models can be used for accurate prediction of workload performance metrics***
- ***Query optimizer can provide features for these models***
- ***Of the shelf models typically sufficient, but may require work to use them properly***
- ***Judicious sampling to collect training data is important***

Tutorial Outline

- Introduction
- Workload-level decisions in database systems
  - Physical design, Scheduling, Progress monitoring, Managing long running queries
- Performance prediction
- ***Break***
- Inter-workload interactions
- Outlook and open problems

Inter-workload Interactions

Inter Workload Interactions



- Positive
- Negative

Negative Workload Interactions

- Workloads W1 and W2 cannot use resource R concurrently
  - CPU, Memory, I/O bandwidth, network bandwidth
- Read-Write issues and the need for transactional guarantees
  - Locking
- Lack of end-to-end control on resource allocation and scheduling for workloads
  - Variation / unpredictability in performance

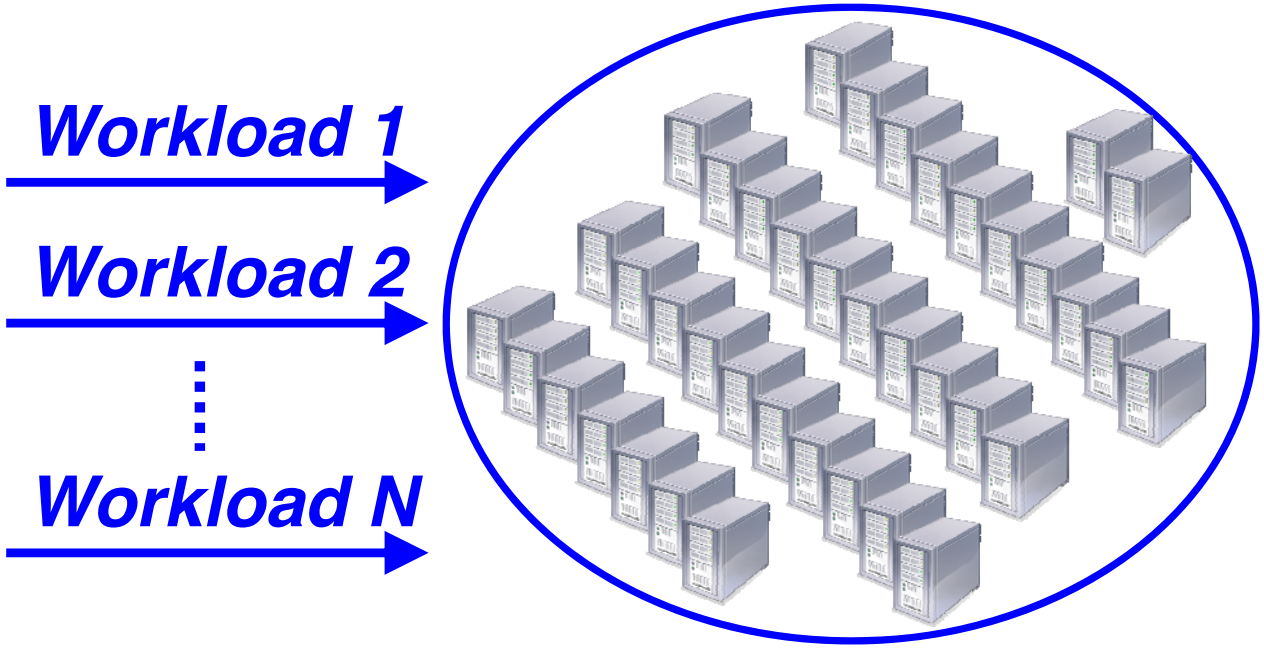
Motivates Workload Isolation

Positive Workload Interactions

- Cross-workload optimizations
  - Multi-query optimizations
  - Scan sharing
  - Caching
  - Materialized views (in-memory)

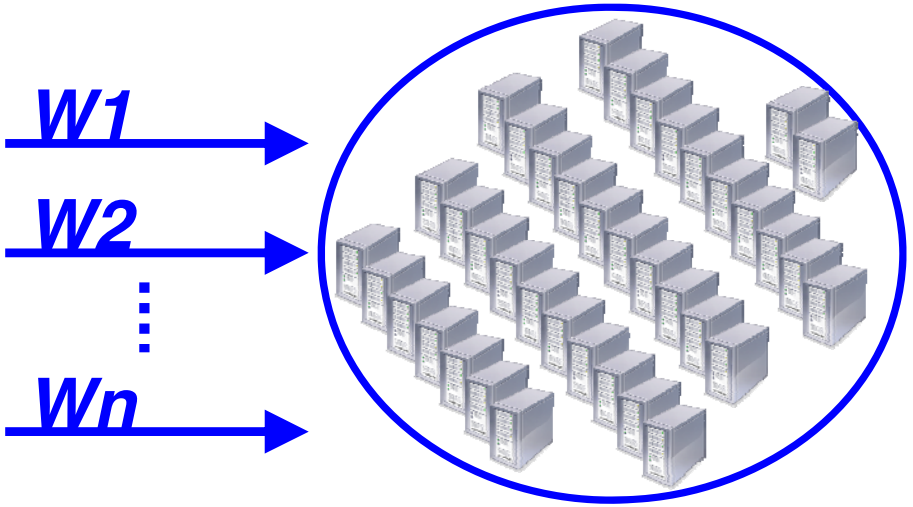
Motivates Shared Execution of Workloads

Inter Workload Interactions



- Research on workload management is heavily biased towards understanding and controlling negative inter-workload Interactions
- Balancing the two types of interactions is an open problem

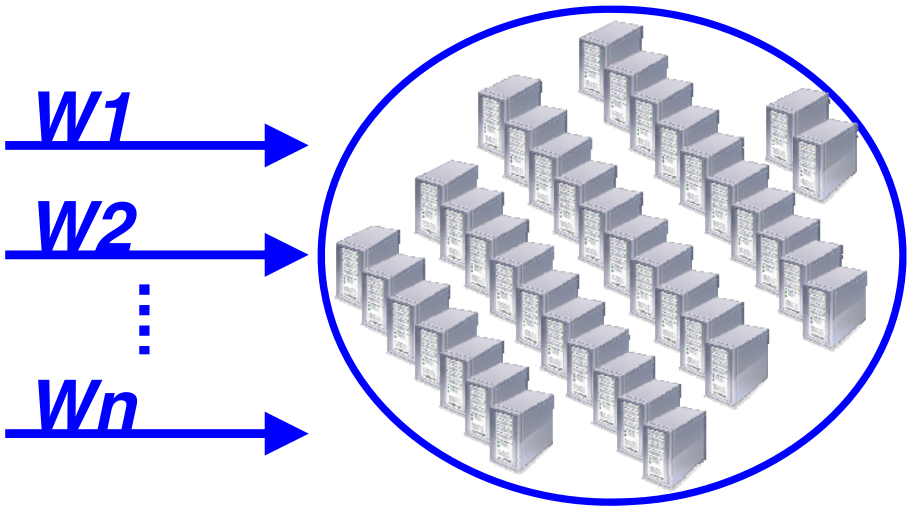
Multi-class Workloads



Kurt P. Brown, Manish Mehta, Michael J. Carey, Miron Livny: Towards Automated Performance Tuning for Complex Workloads, VLDB 1994

- Workload:
  - Multiple user-defined classes. Each class Wi defined by a target average response time
  - "No-goal" class. Best effort performance
- Goal: DBMS should pick <MPL,memory> allocation for each class Wi such that Wi's target is met while leaving the maximum resources possible for the "no goal" class
  - Assumption: Fixed MPL for "no goal" class to 1

Multi-class Workloads



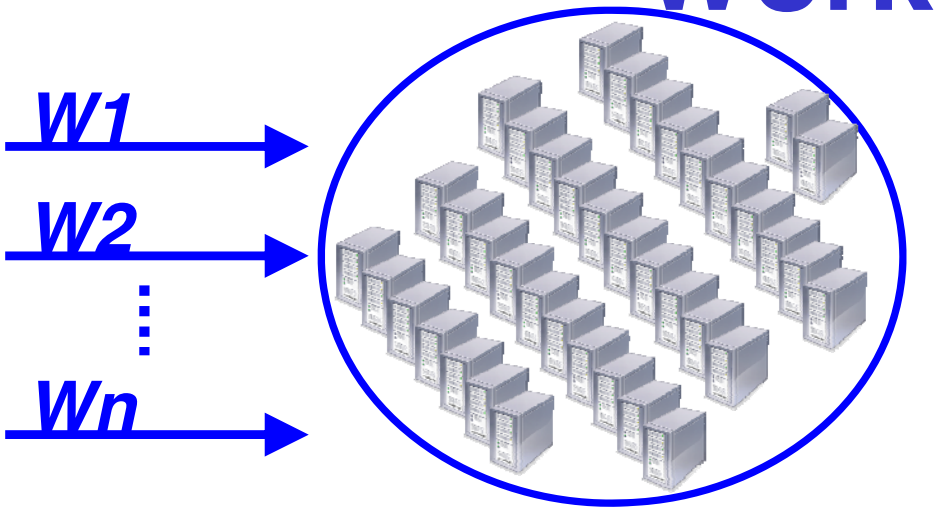
Workload Interdependence:
perf(Wi) = F([MPL],[MEM])

- Assumption: Enough resources available to satisfy requirements of all workload classes
  - Thus, system never forced to sacrifice needs of one class in order to satisfy needs of another
- They model relationship between MPL and Memory allocation for a workload
  - Shared Memory Pool per Workload = Heap + Buffer Pool
  - Same performance can be given by multiple <MPL,Mem> choices

Multi-class Workloads

- Heuristic-based per-workload feedback-driven algorithm
  - M&M algorithm
- Insight: Best return on consumption of allocated heap memory is when a query is allocated either its maximum or its minimum need [Yu and Cornell, 1993]
- M&M boils down to setting three knobs per workload class:
  - maxMPL: queries allowed to run at max heap memory
  - minMPL: queries allowed to run at min heap memory
  - Memory pool size: Heap + Buffer pool

Real-time Multi-class Workloads



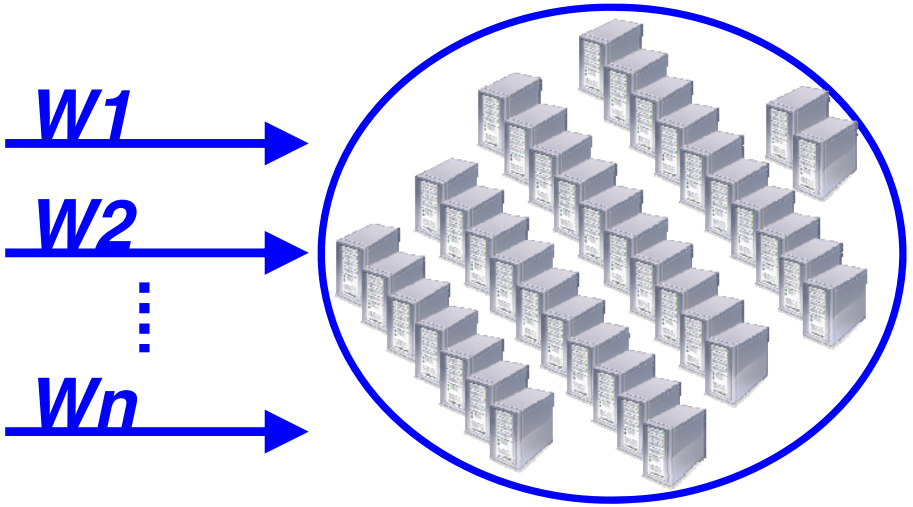
HweeHwa Pang, Michael J. Carey, Miron Livny: Multiclass Query Scheduling in Real-Time Database Systems. IEEE TKDE 1995

- Workload: Multiple user-defined classes
  - Queries come with deadlines, and each class Wi is defined by a miss ratio (% of queries that miss their deadlines)
  - DBA specifies **miss distribution**: how misses should be distributed among the classes

Real-time Multi-class Workloads

- Feedback-driven algorithm called Priority Adaptation Query Resource Scheduling
- MPL and Memory allocation strategies are similar in spirit to the M&M algorithm
- Queries in each class are divided into two Priority Groups: Regular and Reserve
  - Queries in Regular group are assigned a priority based on their deadlines (Earliest Deadline First)
  - Queries in Reserve group are assigned a lower priority than those in Regular group
- Miss ratio distribution is controlled by adjusting size of regular group across workload classes

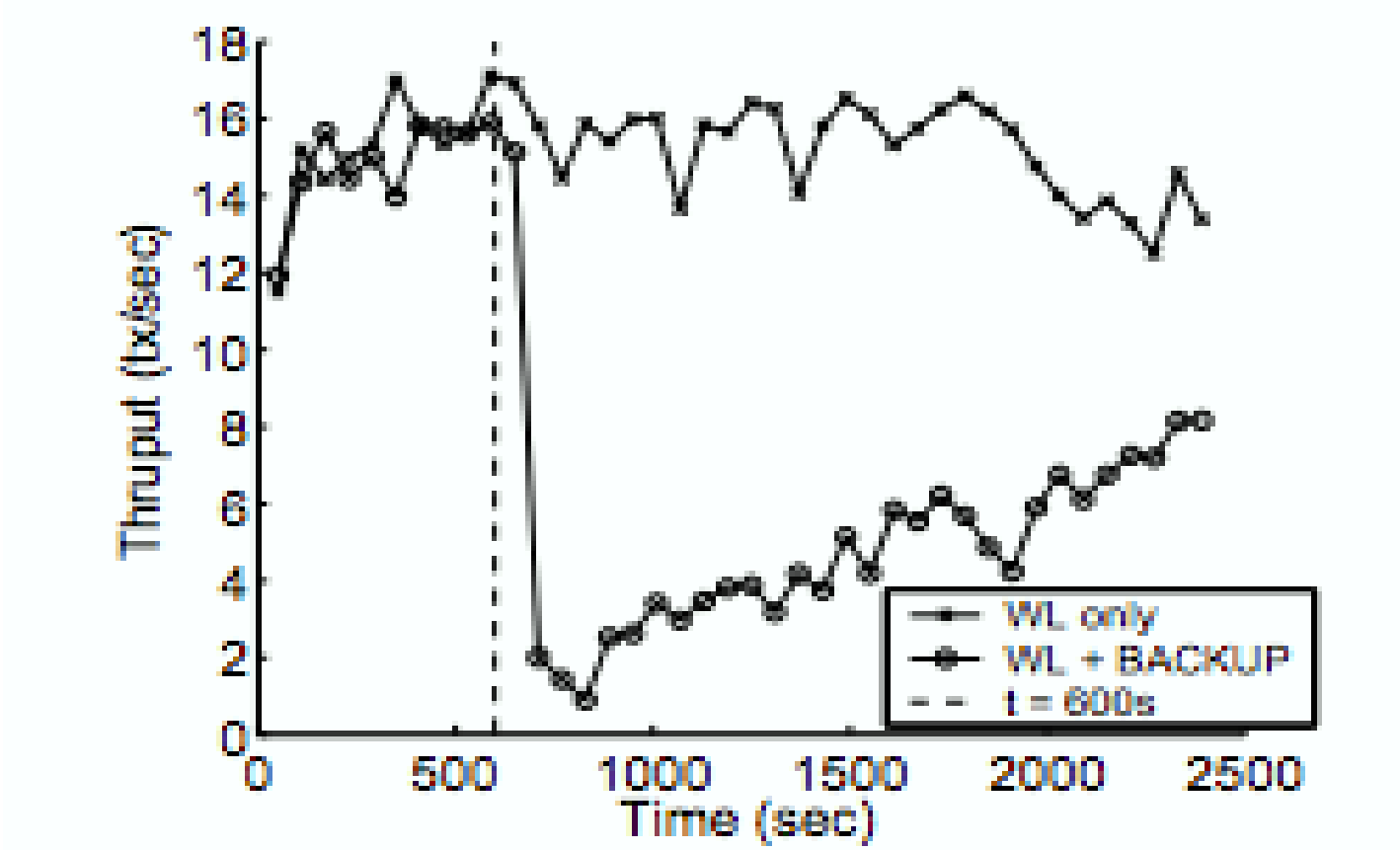
Throttling System Utilities



Sujay S. Parekh, Kevin Rose, Joseph L. Hellerstein, Sam Lightstone, Matthew Huras, Victor Chang: Managing the Performance Impact of Administrative Utilities. DSOM 2003

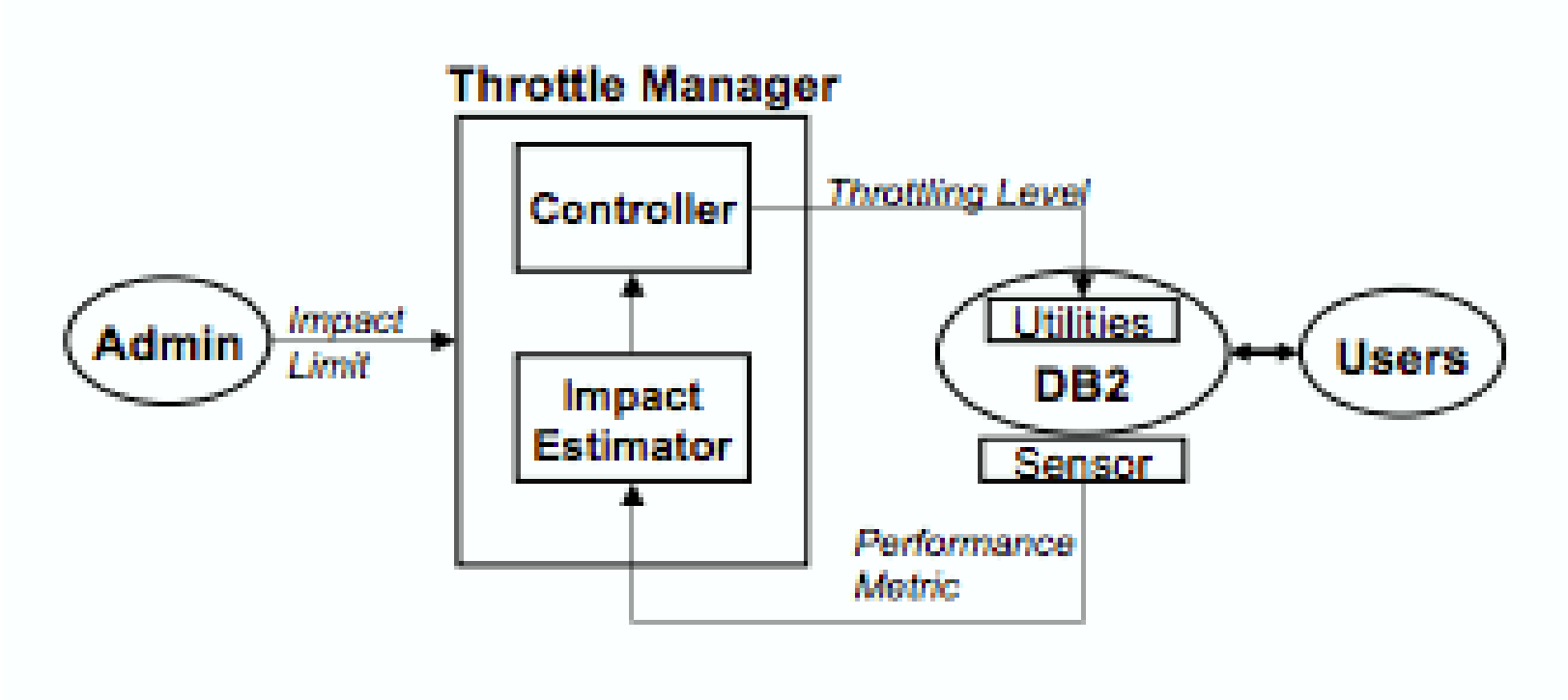
- Workload: Regular DBMS processing Vs. DBMS system utilities like backups, index rebuilds, etc.

Throttling System Utilities



- DBA should be able to say: have no more than x% performance degradation of the production work as a result of running system utilities

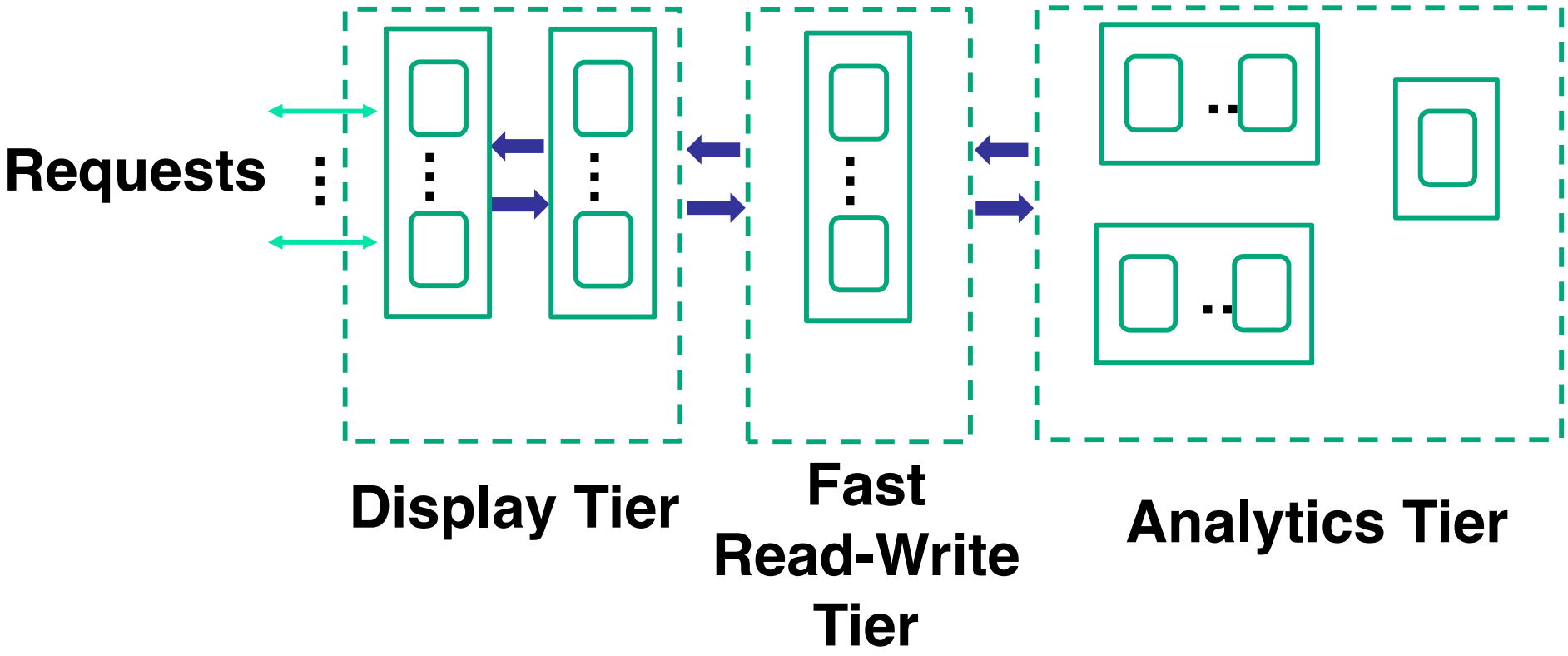
Throttling System Utilities



- Control theoretic approach to make utilities sleep
- Proportional-Integral controller from linear control theory

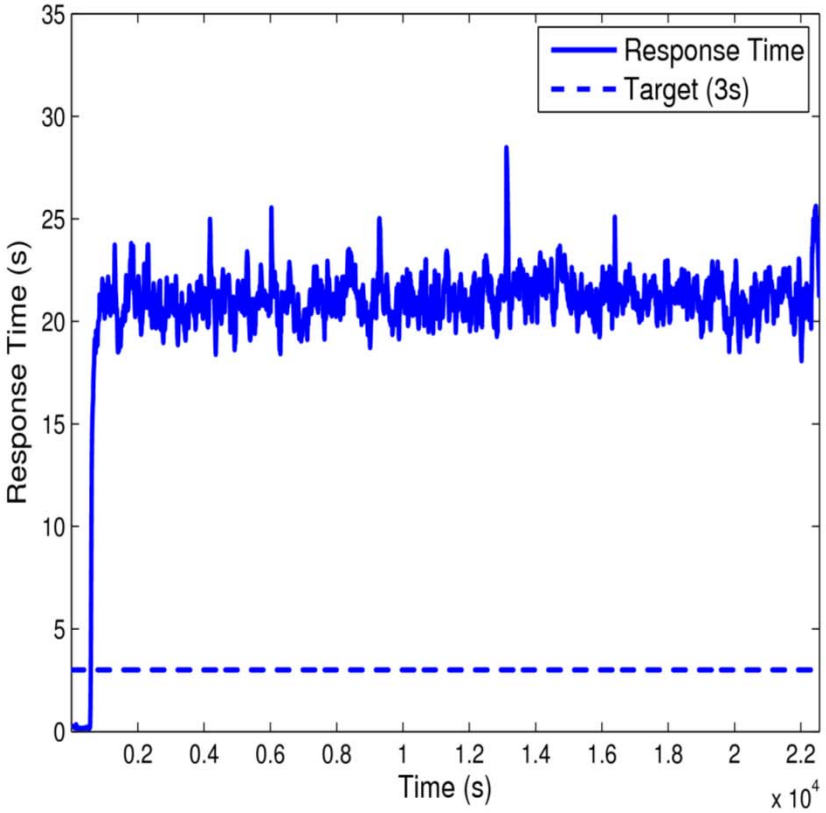
Elasticity in Key-Value Stores

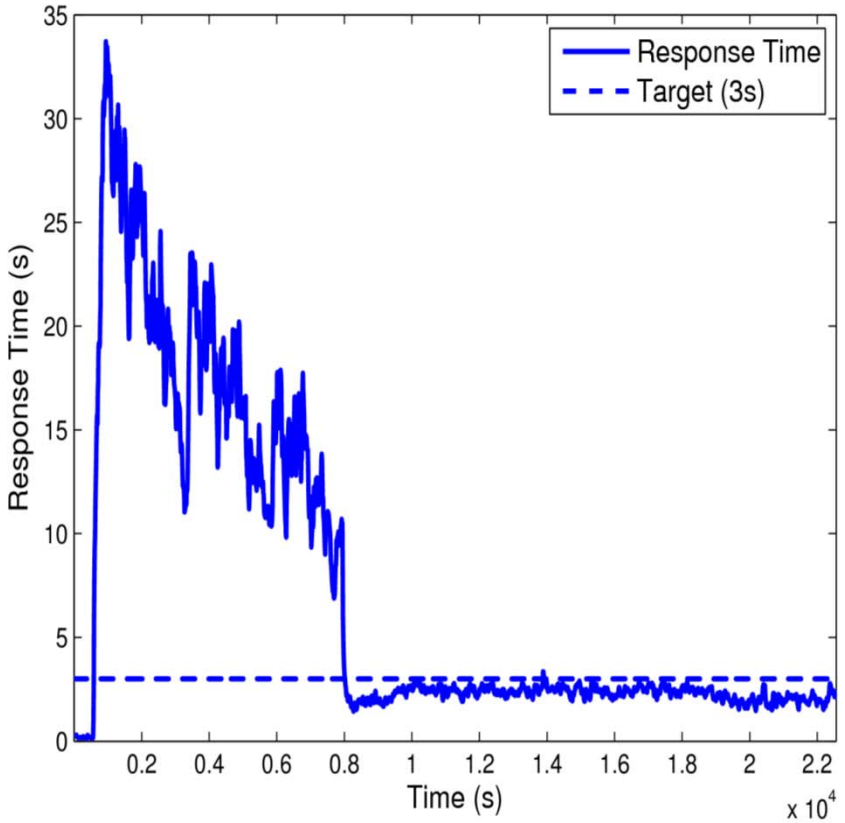
- Modern three-tier data-intensive services
- Each with different workloads and responsibility

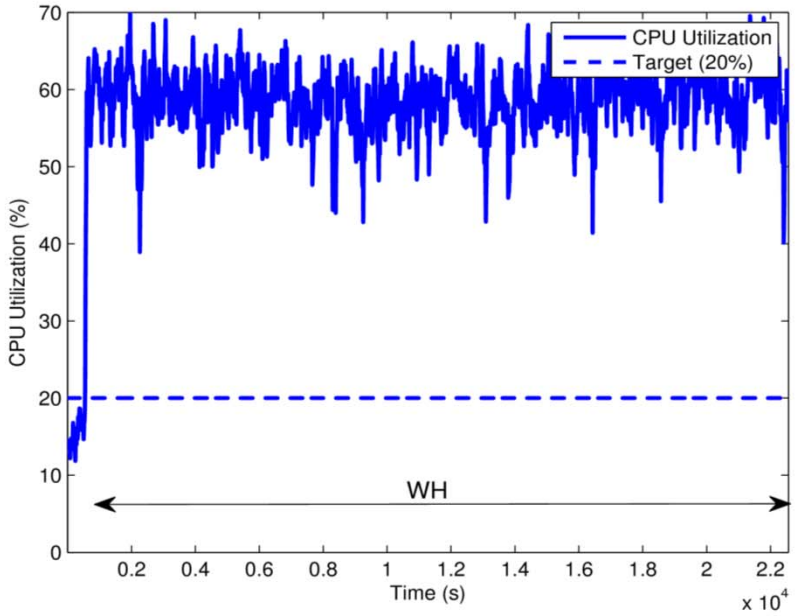


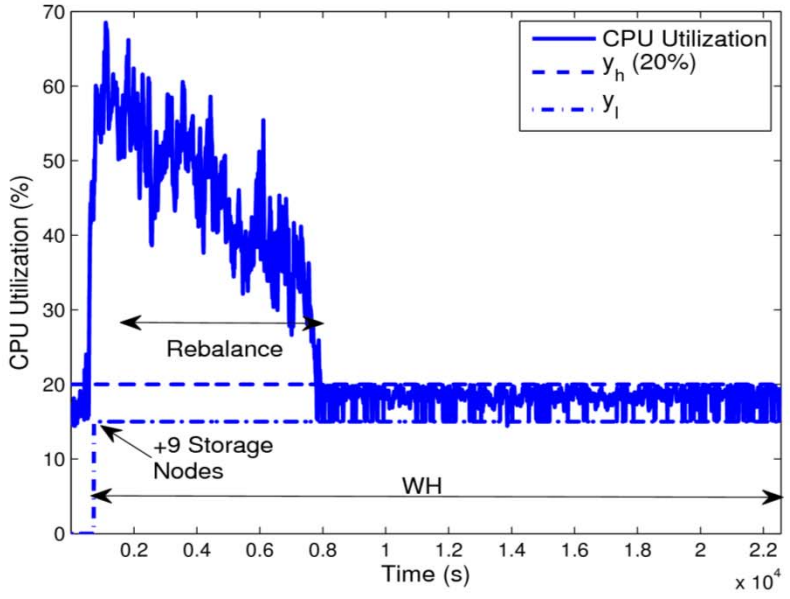
Elasticity in Key-Value Stores

- Opportunity for elasticity – acquire and release servers in response to dynamic workloads to ensure requests are served within acceptable latency
- Challenges:
  - Cloud providers allocate resources in discrete units
  - Data rebalancing – need to move data before getting performance benefits
  - Interference to workloads (requests) – Uses the same resources (I/O) to serve requests
  - Actuator delays – there is delay before improvements


Response Time
Target (3s)
Response Time (s)
Time (s)
x 10^4


Response Time
Target (3s)
Response Time (s)
Time (s)
x 10^4


CPU Utilization
Target (20%)
CPU Utilization (%)
WH
Time (s)
x 10^4


CPU Utilization
y_h (20%)
y_l
CPU Utilization (%)
Rebalance
+9 Storage Nodes
WH
Time (s)
x 10^4

Elasticity in Key-Value Stores

Harold Lim, Shivnath Babu, and Jeffrey Chase. "Automated Control for Elastic Storage." *ICAC*, 2010.

- Describes the Elastore system
- Elastore is composed of *Horizontal Scale Controller (HSC)* for provisioning nodes, *Data Rebalance Controller (DRC)* for controlling data transfer between nodes, and a *State Machine* for coordinating HSC and DRC

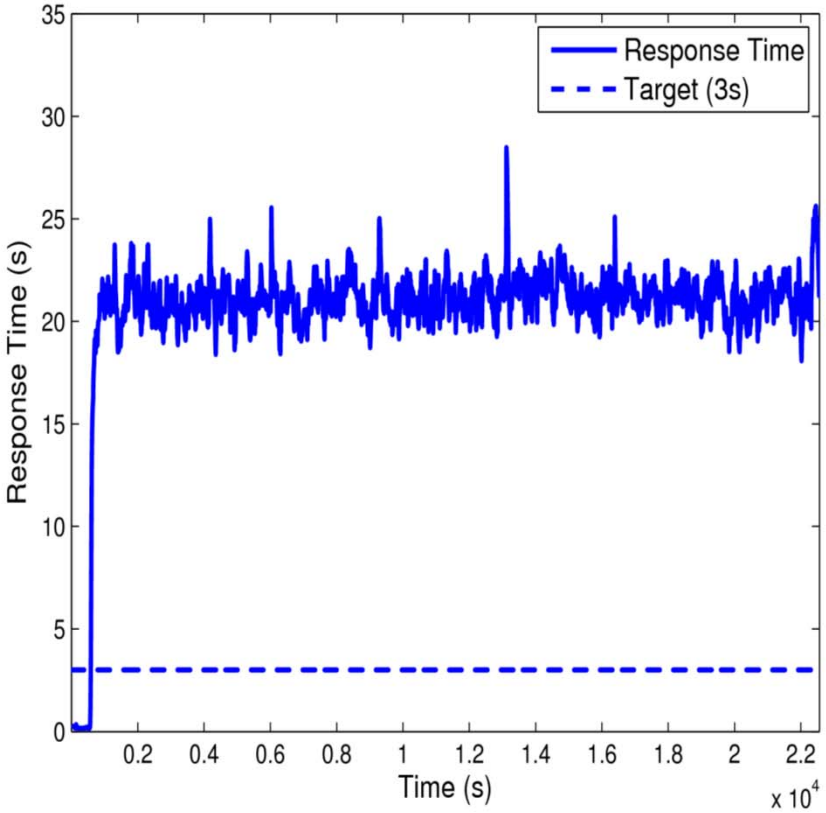
Horizontal Scale Controller

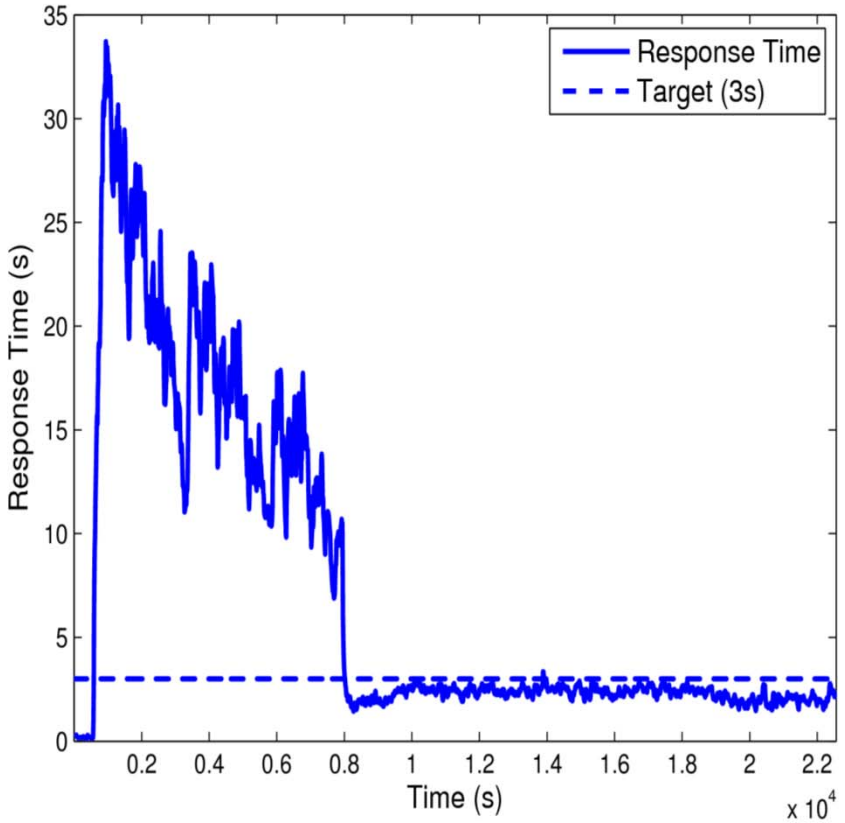
- Control Policy: *proportional thresholding* to control cluster size, with average CPU as sensor
  - Modifies classical integral control to have a dynamic target range (dependent on the size of the cluster)
  - Prevents oscillations due to discrete/coarse actuators
  - Ensures efficient use of resources

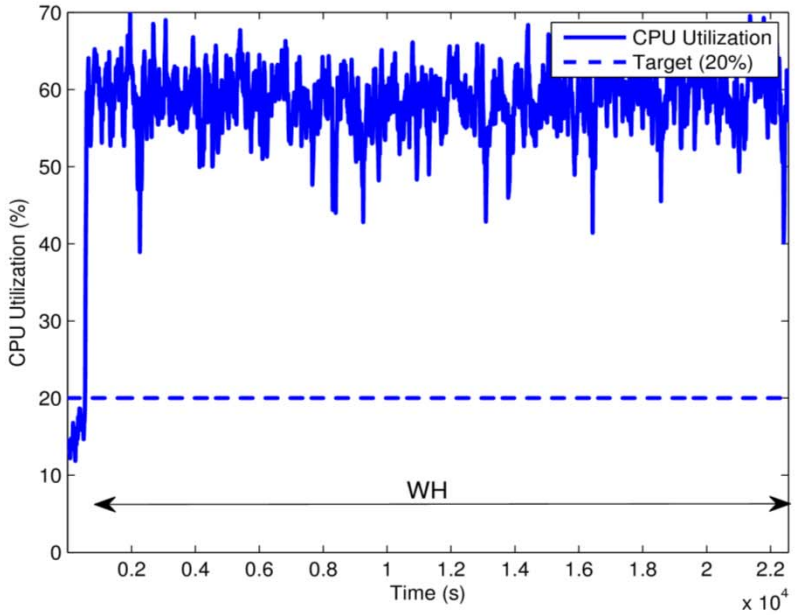
$$u_{k+1} = \begin{cases} u_k + K_i \times (y_h - y_k) & \text{if } y_h < y_k \\ u_k + K_i \times (y_l - y_k) & \text{if } y_l > y_k \\ u_k & \text{otherwise} \end{cases}$$

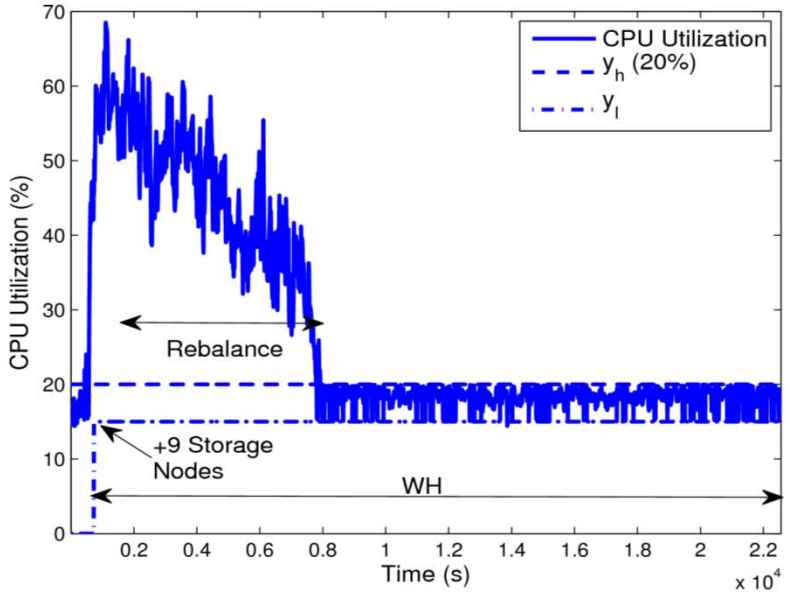
Data Rebalance Controller

- Controls the bandwidth *b* allocated to rebalance
  - The maximum amount of bandwidth each node can devote to rebalancing
  - The choice of b affects the tradeoff between lag (time to completion of rebalancing) and interference (performance impact on workload)
  - Modeled the time to completion as a function of bandwidth and size of data
  - Modeled interference as a function of bandwidth and per-node workload
  - Choice of b is posed as a cost-based optimization problem


Response Time
Target (3s)
Response Time (s)
Time (s)
x 10^4

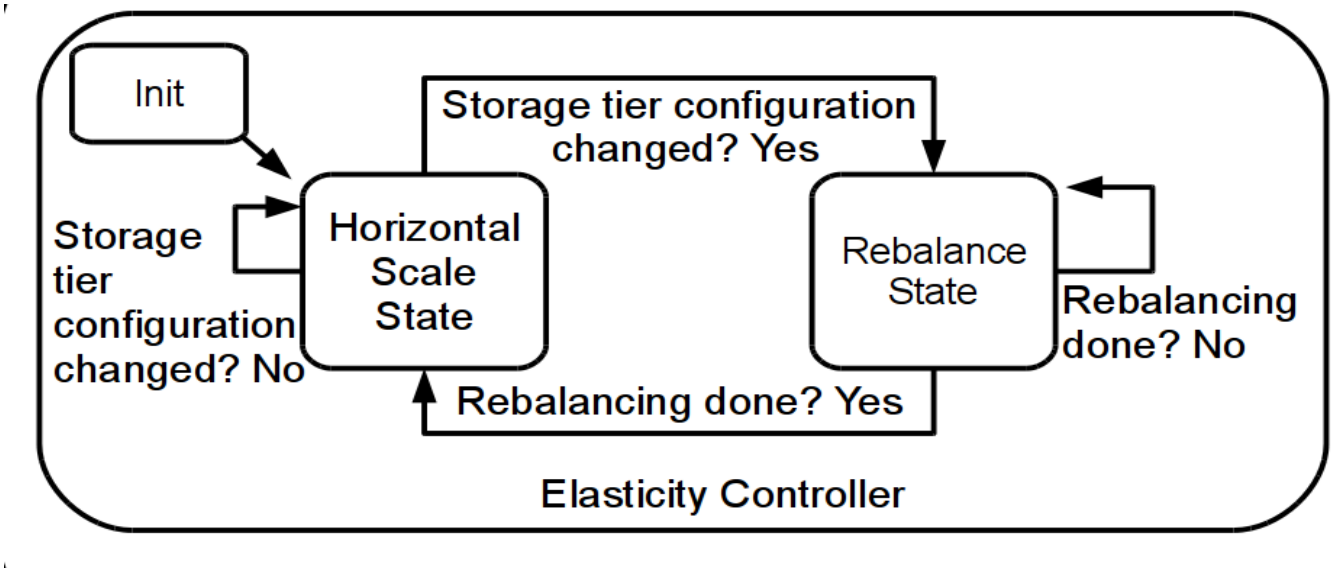

Response Time
Target (3s)
Response Time (s)
Time (s)
x 10^4


CPU Utilization
Target (20%)
CPU Utilization (%)
WH
Time (s)
x 10^4


CPU Utilization
y_h (20%)
y_l
CPU Utilization (%)
Rebalance
+9 Storage Nodes
WH
Time (s)
x 10^4

State Machine

- Manages the mutual dependencies between HSC and DRC
  - Ensures the controller handles DRC's actuator lag
  - Ensures interference and sensor noise introduced by rebalancing does not affect the HSC



Impact of Long-Running Queries

Stefan Krompass, Harumi Kuno, Janet L. Wiener, Kevin Wilkinson, Umeshwar Dayal, Alfons Kemper. "Managing Long-Running Queries." *EDBT*, 2009.

- Heavy Vs. Hog
- Overload and Starving

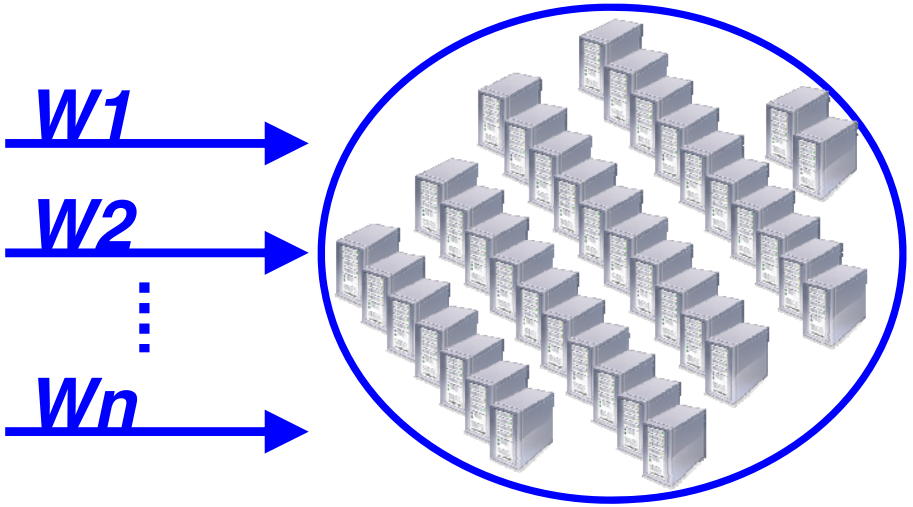
	Query expected to be long	Query progress reasonable	Uses equal share of resources
expected-heavy	Yes	Yes	Equal share
expected-hog	Yes	Yes	> Equal share
surprise-heavy	No	Yes	Equal share
surprise-hog	No	Yes	> Equal share
overload	No	No	Equal share
starving	No	No	< Equal share

Impact of Long-Running Queries

- Commercial DBMSs give rule-based languages for the DBAs to specify the actions to take to deal with "problem queries"
- However, implementing good solutions is an art
  - How to quantify progress? How to attribute resource usage to queries? How to distinguish an overloaded scenario from a poorly-tuned scenario? **How to connect workload management actions with business importance?**

	Query expected to be long	Query progress reasonable	Uses equal share of resources
expected-heavy	Yes	Yes	Equal share
expected-hog	Yes	Yes	> Equal share
surprise-heavy	No	Yes	Equal share
surprise-hog	No	Yes	> Equal share
overload	No	No	Equal share
starving	No	No	< Equal share

Utility Functions



Baoning Niu, Patrick Martin, Wendy Powley, Paul Bird, Randy Horman: Adapting Mixed Workloads to Meet SLOs in Autonomic DBMSs, SMDB 2007

- Workload: Multiple user-defined classes. Each class has:
  - Performance target(s)
  - Business importance
- Designs utility functions that quantify the utility obtained from allocating more resources to each class
  - Gives an optimization objective
  - Implemented over IBM DB2's Query Patroller

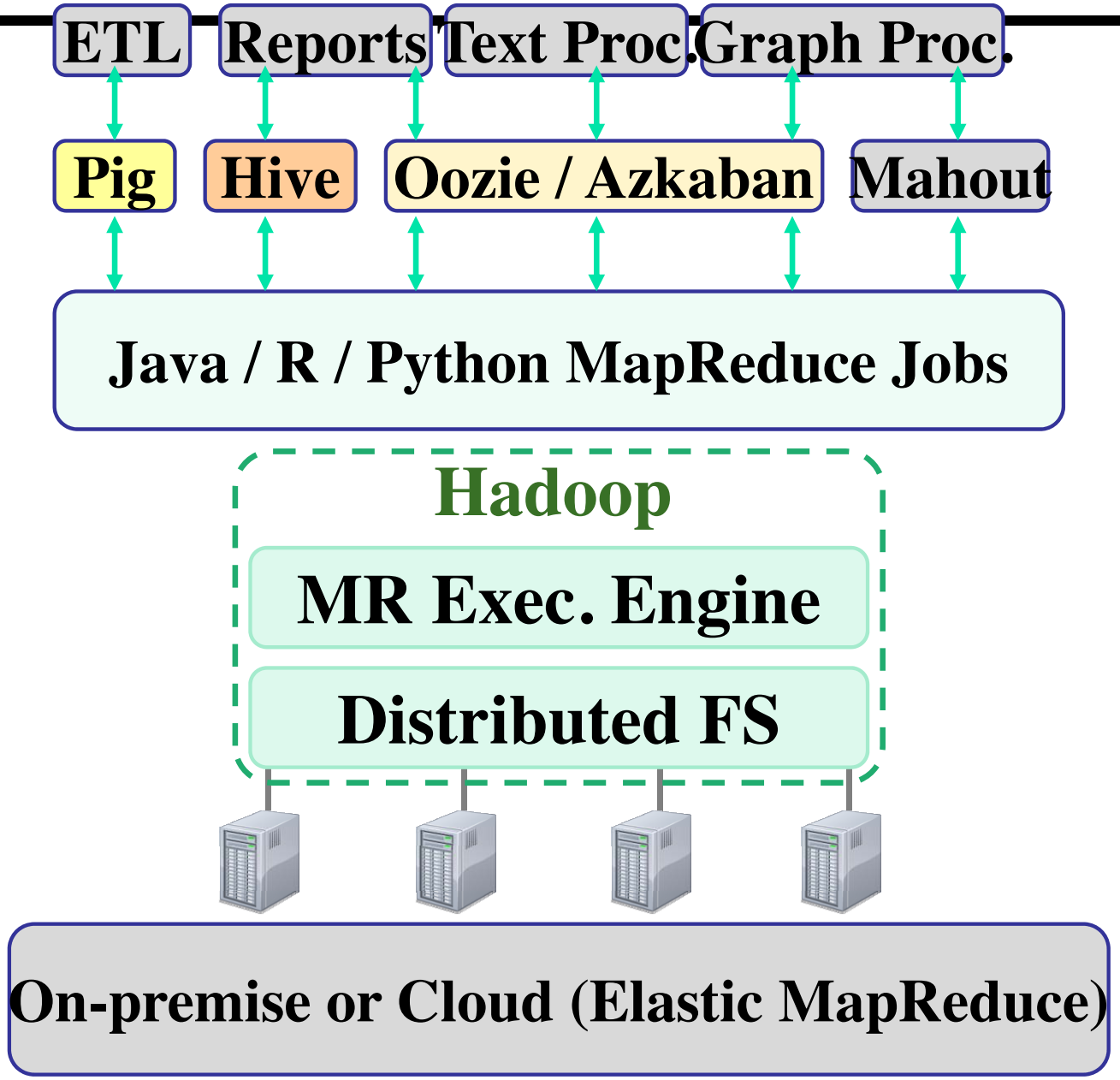
Seminar Outline

- Introduction
- Workload-level decisions in database systems
  - Physical design
  - Progress monitoring
  - Managing long running queries
- Performance prediction
- Progress Monitoring
- Inter workload interactions
- Outlook and Open Problems

On to MapReduce systems

DBMS Vs. MapReduce (MR) Stack

- Narrow waist of the MR stack
- Workload mgmt. done at the level of MR jobs

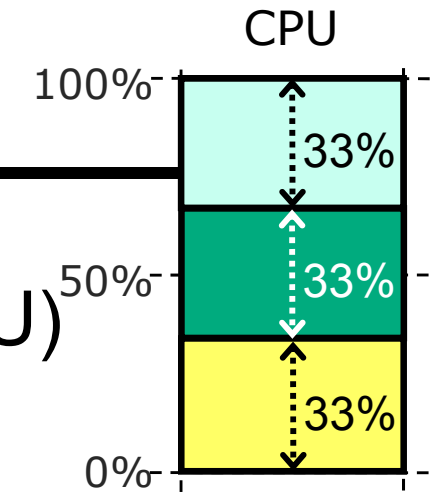


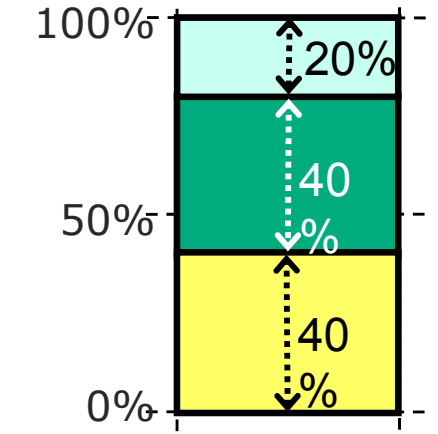
MapReduce Workload Mgmt.

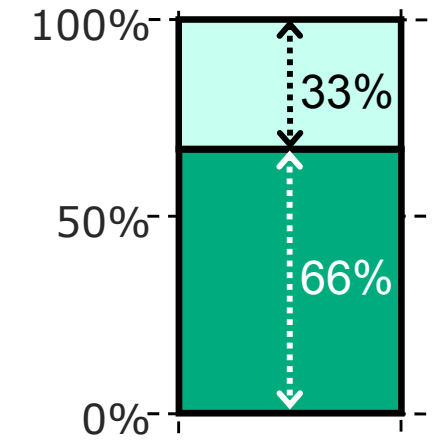
- Resource management policy: Fair sharing
- Unidimensional fair sharing
  - Hadoop's Fair scheduler
  - Dryad's Quincy scheduler
- Multi-dimensional fair sharing
- Resource management frameworks
  - Mesos
  - Next Generation MapReduce (YARN)
  - Serengeti

What is Fair Sharing?

- n users want to share a resource (e.g., CPU)
  - Solution: Allocate each 1/n of the resource
- *Generalized by max-min fairness*
  - Handles if a user wants less than her fair share
  - E.g., user 1 wants no more than 20%
- Generalized by *weighted max-min fairness*
  - Give weights to users according to importance
  - User 1 gets weight 1, user 2 weight 2



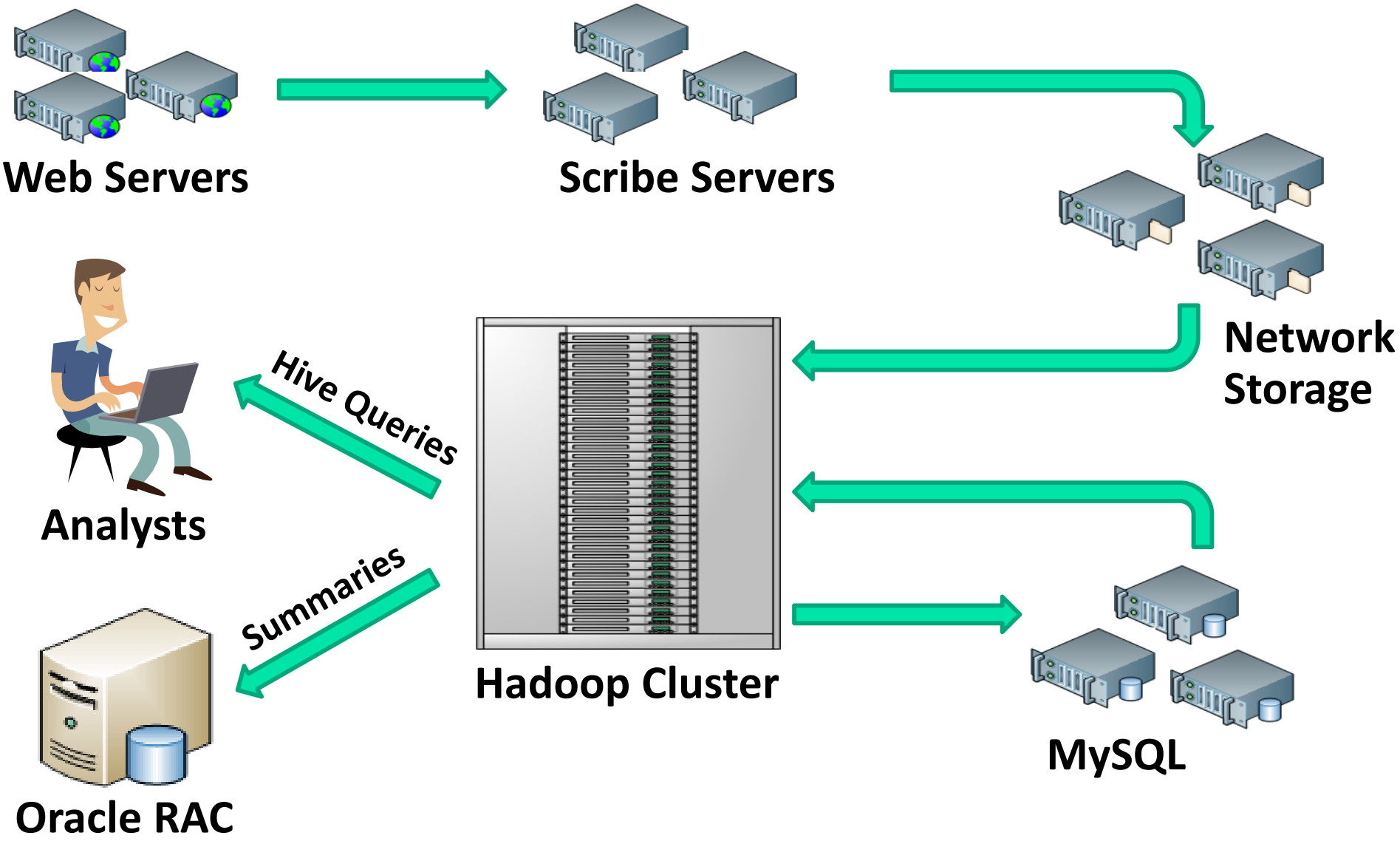




Why Care about Fairness?

- Desirable properties of max-min fairness
  - *Isolation policy:*
    - A user gets her fair share irrespective of the demands of other users
    - Users cannot affect others beyond their fair share
  - *Flexibility separates mechanism from policy:*
    Proportional sharing, priority, reservation, ...
- *Many schedulers* use max-min fairness
  - Datacenters: Hadoop's Fair Scheduler, Hadoop's Capacity Scheduler, Dryad's Quincy
  - OS: rr, prop sharing, lottery, linux cfs, ...
  - Networking: wfq, wf2q, sfq, drr, csfq, ...

Example: Facebook Data Pipeline

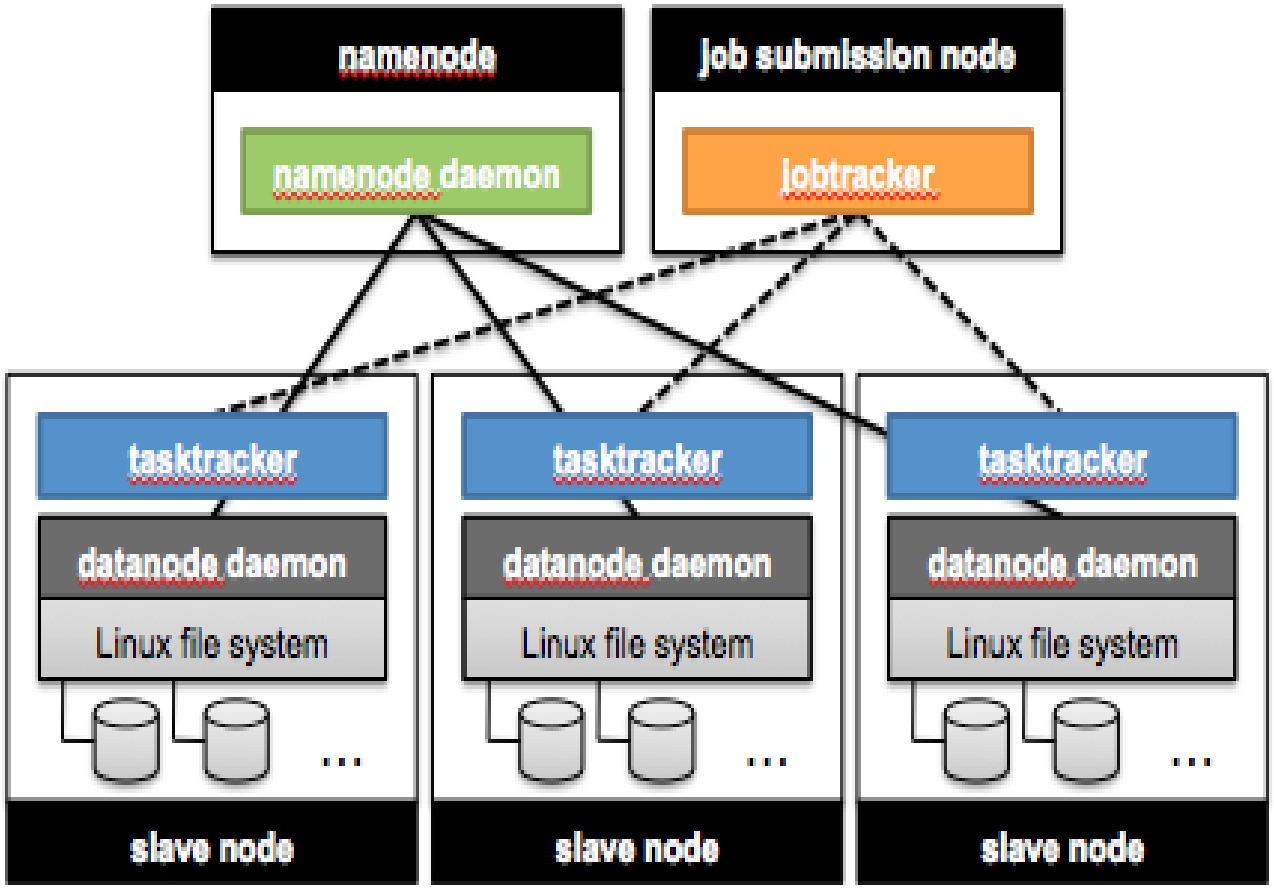


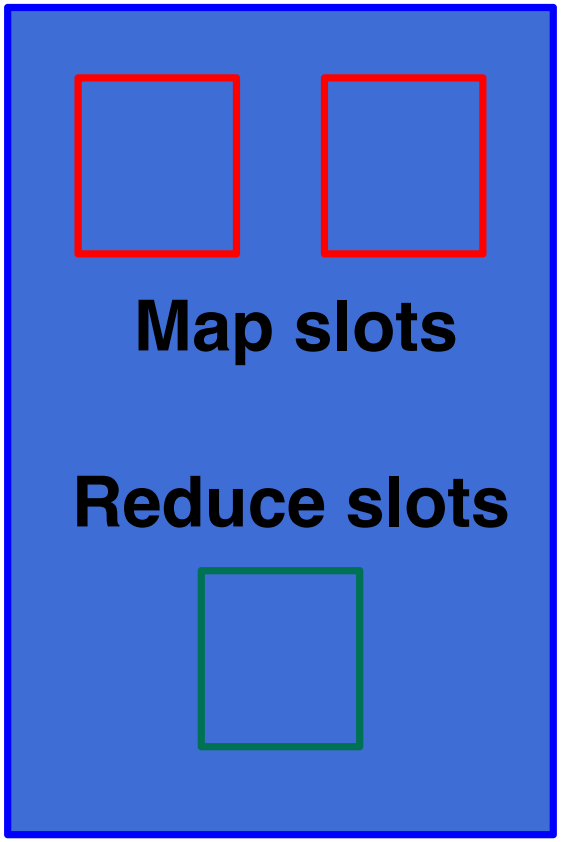
Example: Facebook Job Types

- **Production jobs:** load data, compute statistics, detect spam, etc.
- **Long experiments:** machine learning, etc.
- **Small ad-hoc queries:** Hive jobs, sampling

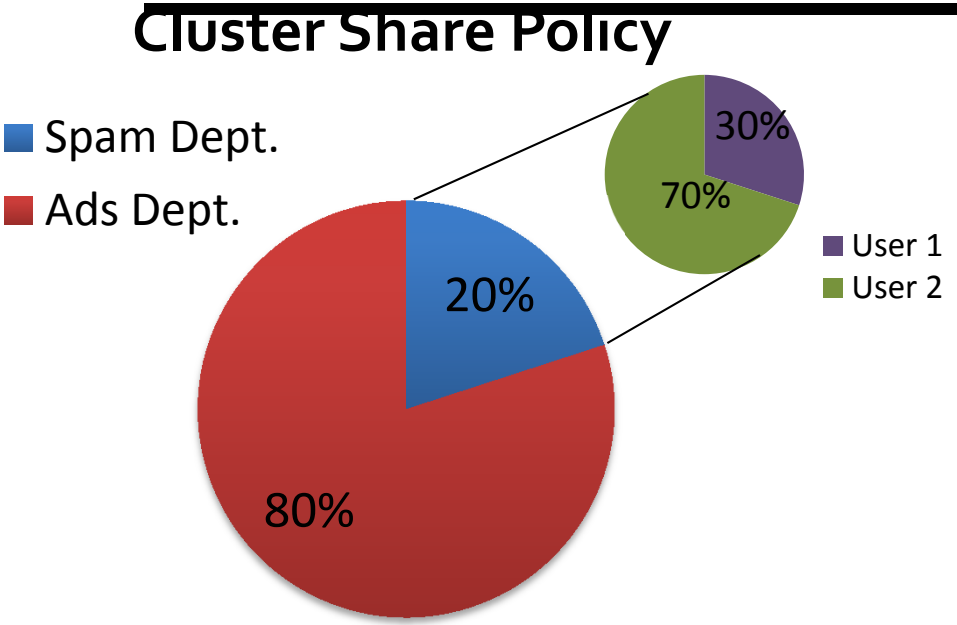
GOAL: Provide fast response times for small jobs and guaranteed service levels for production jobs

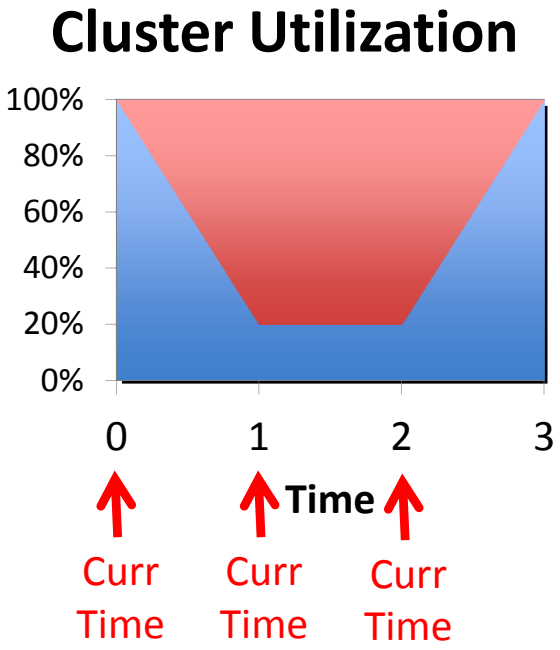
Task Slots in Hadoop

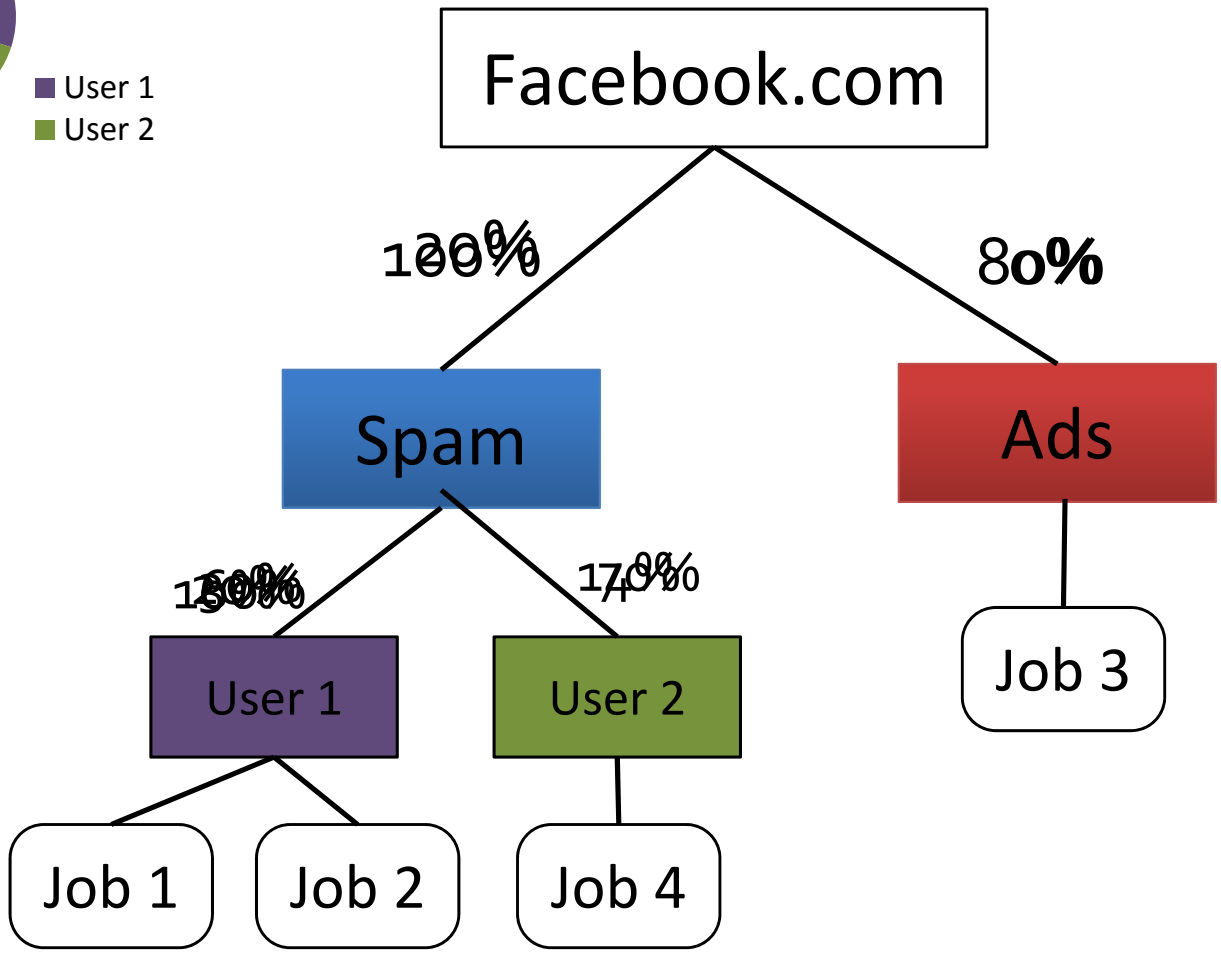




Example: Hierarchical Fair Sharing







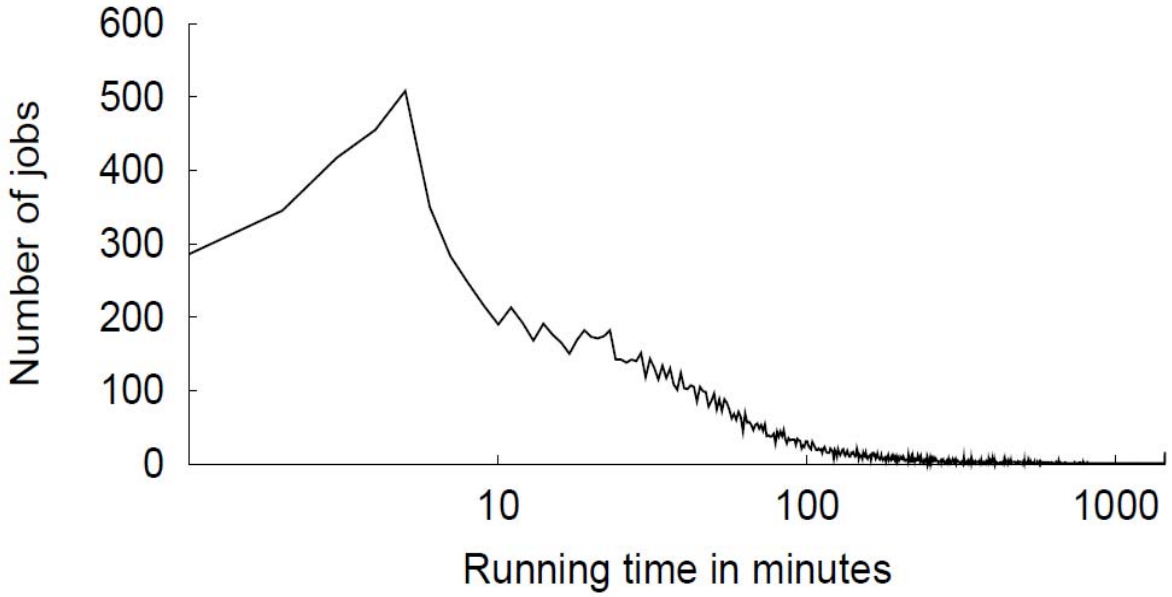
Hadoop's Fair Scheduler

M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, Job Scheduling for Multi-User MapReduce Clusters, UC Berkeley Technical Report UCB/EECS-2009-55, April 2009

- Group jobs into "*pools*" *each with* a guaranteed *minimum share*
  - Divide each pool's minimum share among its jobs
  - Divide excess capacity among all pools
- When a task slot needs to be assigned:
  - If there is any pool below its min share, schedule a task from it
  - Else pick a task from the pool we have been most unfair to

Quincy: Dryad's Fair Scheduler

Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, Andrew Goldberg: Quincy: fair scheduling for distributed computing clusters. SOSP 2009

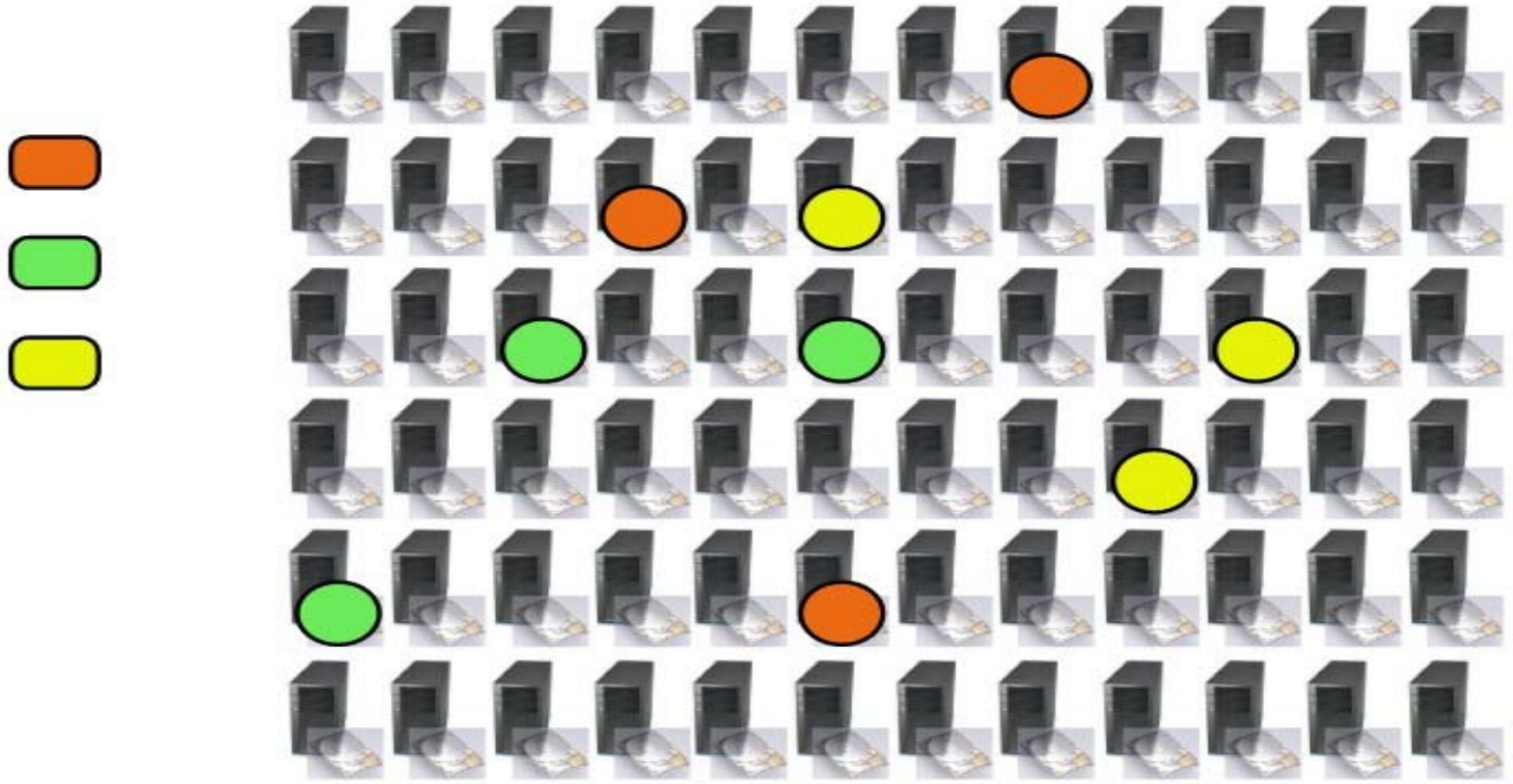


Run time(m)	5	10	15	30	60	300
% Jobs	18.9	28.0	34.7	51.31	72.0	95.7

Goals in Quincy

- Fairness: If a job takes t time when run alone, and J jobs are running, then the job should take no more time than Jt
- Sharing: Fine-grained sharing of the cluster, minimize idle resources (maximize throughput)
- Maximize data locality

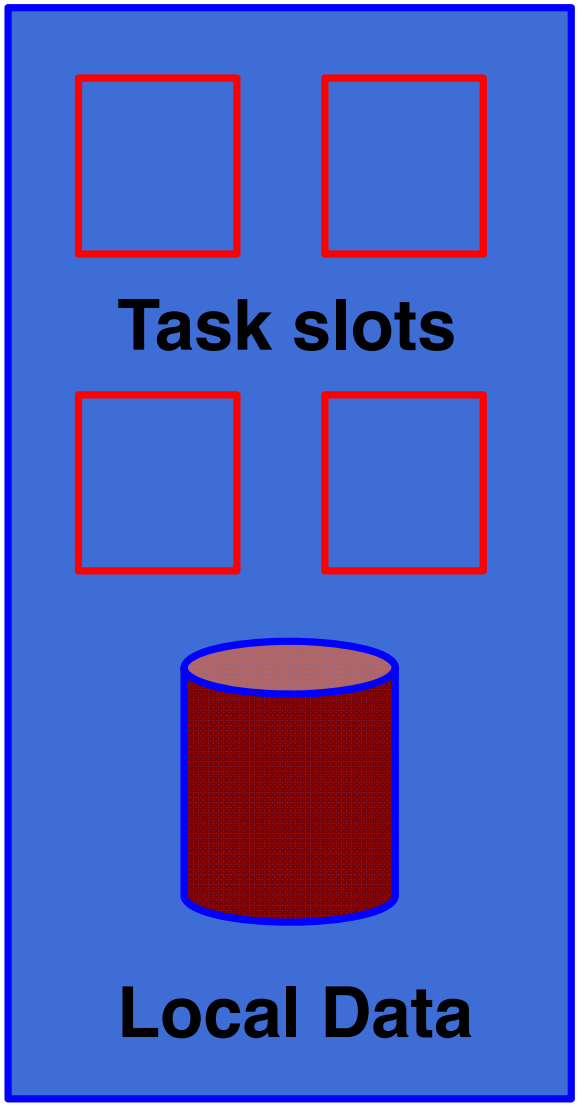
Data Locality



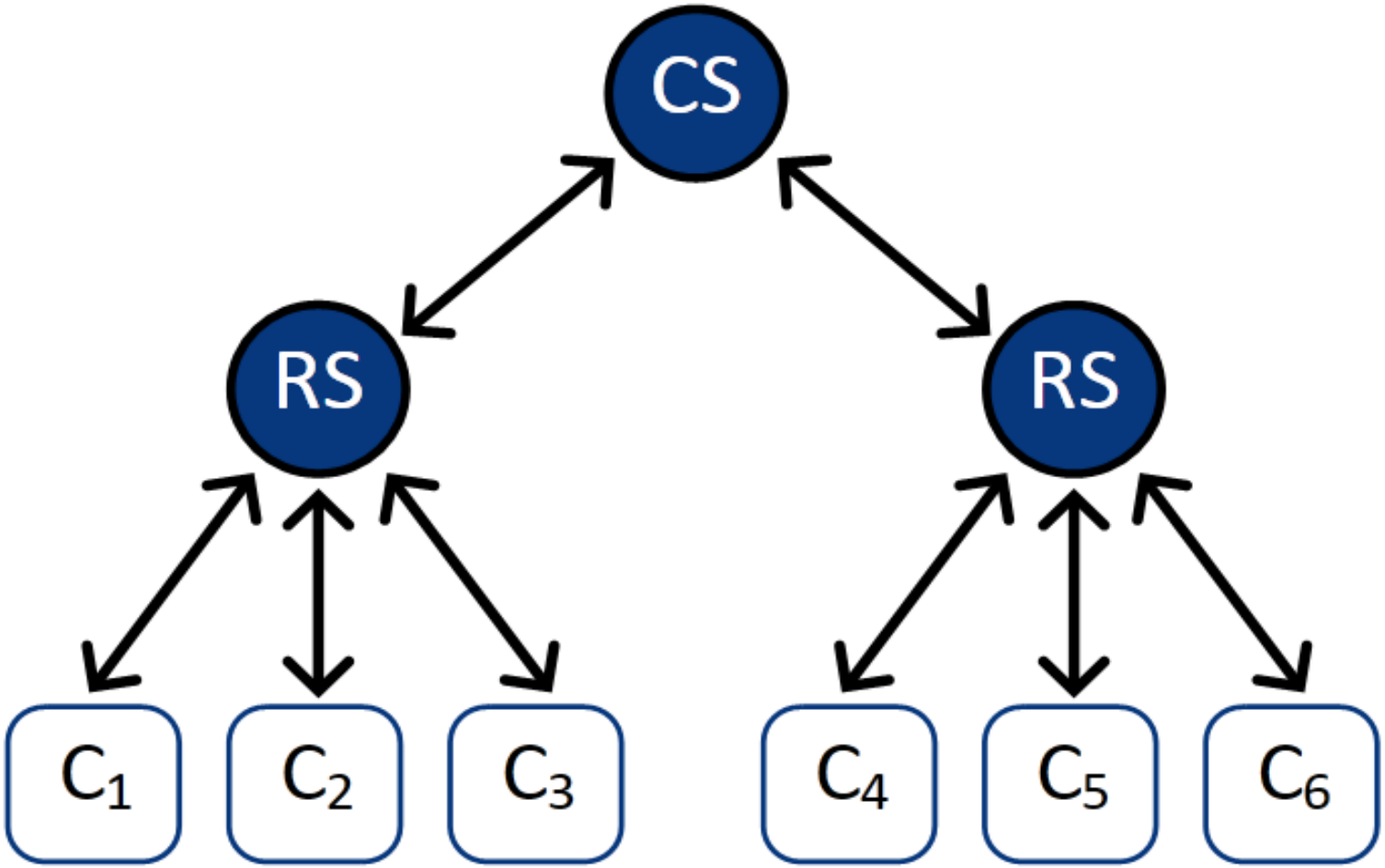
- Data transfer costs depend on where data is located

Goals in Quincy

- Fairness: If a job takes t time when run alone, and J jobs are running, then the job should take no more time than Jt
- Sharing: Fine-grained sharing of the cluster, minimize idle resources (maximize throughput)
- Maximize data locality
- Admission control to limit to K concurrent jobs
  - choice trades off fairness wrt locality and avoiding idle resources
- Assumes fixed task slots per machine



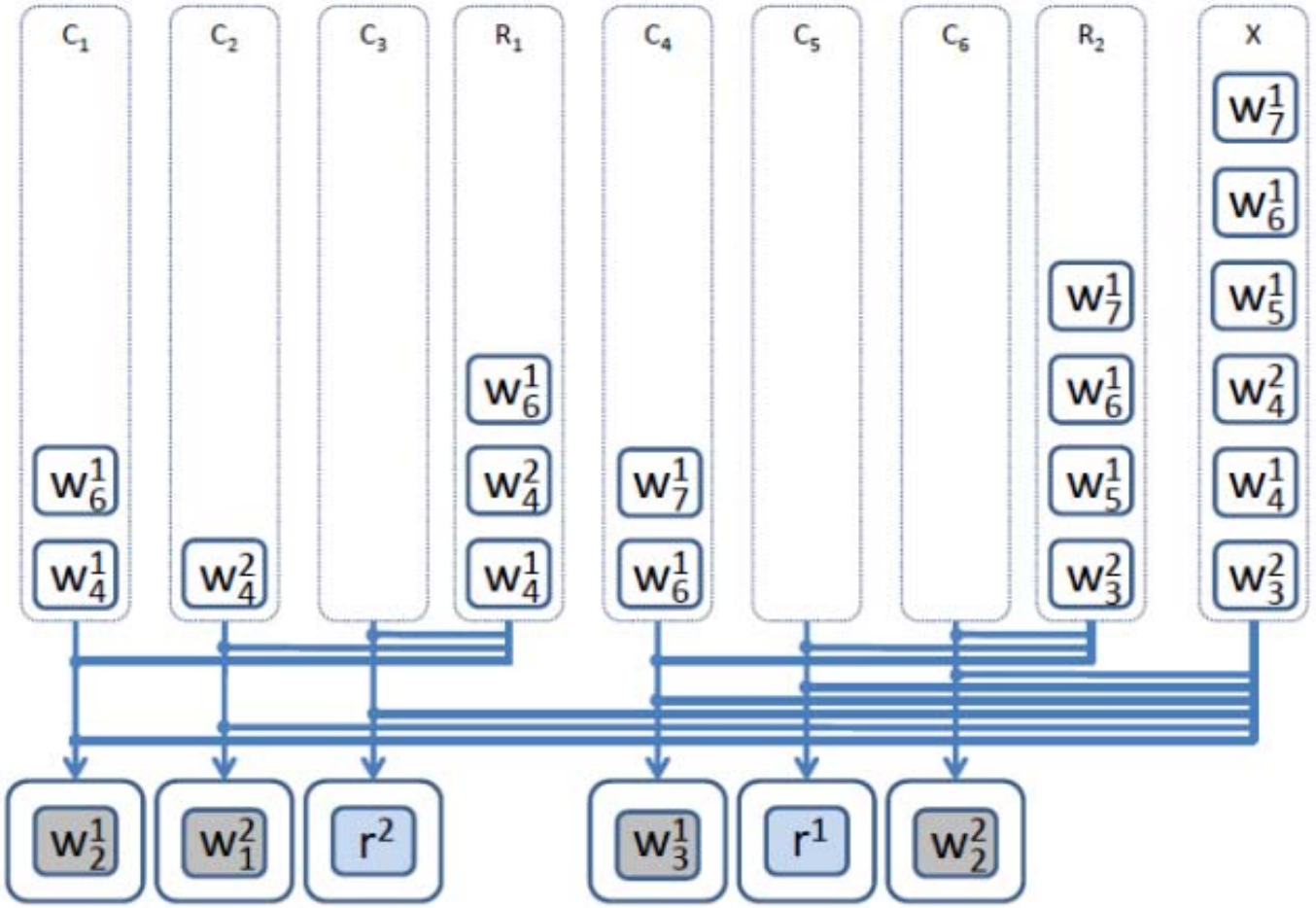
Queue-based Vs. Graph-based Scheduling



Cluster Architecture

Queue-based Vs. Graph-based Scheduling

Queues

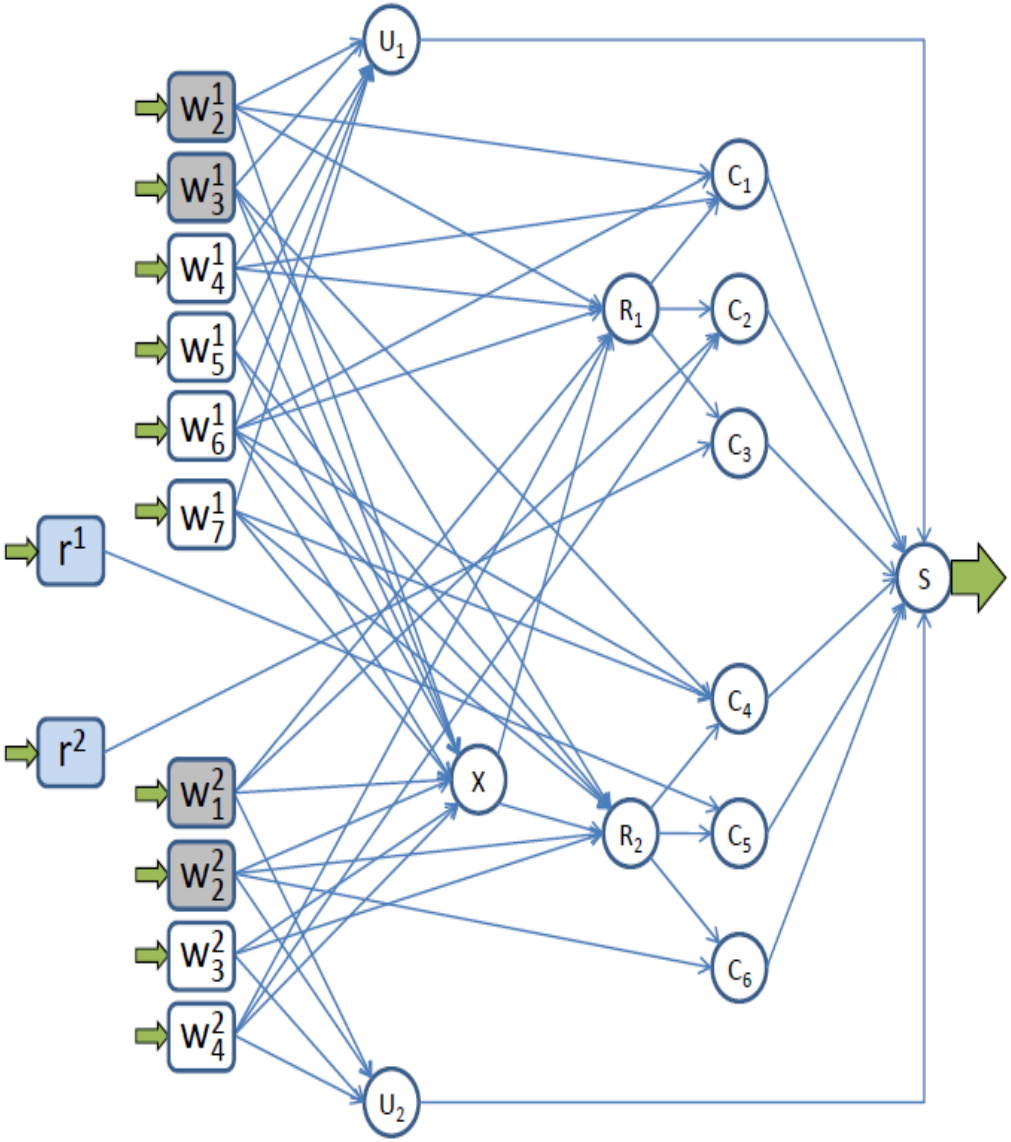


Queue-Based Scheduling

- Greedy (G):
  - Locality-based preferences
  - Does not consider fairness
- Simple Greedy Fairness (GF):
  - “block” any job that has its fair allocation of resources
  - Schedule tasks only from unblocked jobs
- Fairness with preemption (GFP):
  - The over-quota tasks will be killed, with shorter-lived ones killed first
- Other policies

Graph-based Scheduling

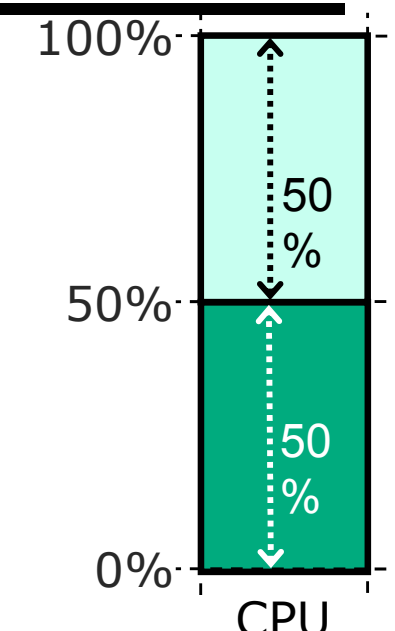
- Encode cluster structure, jobs, and tasks as a flow network
  - Captures entire state of system at any point of time
- Edge costs encode policy
  - cost of waiting (not being scheduled yet)
  - cost of data transfers
- Solving the min-cost flow problem gives a scheduling assignment



However

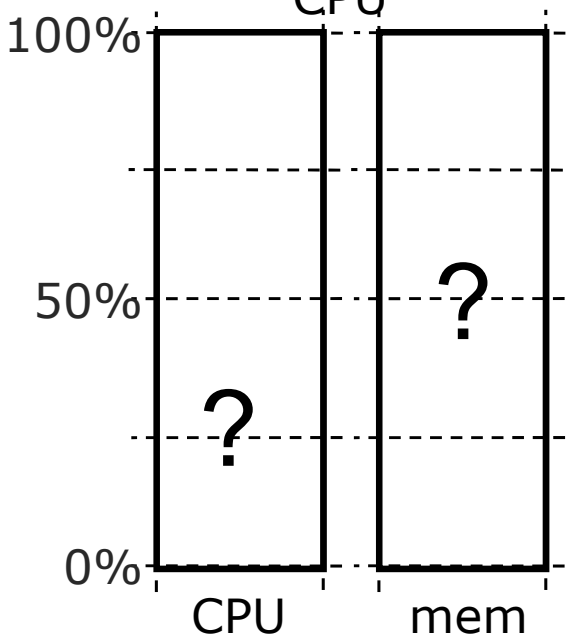
Single resource example

- 1 resource: CPU
- User 1 wants <1 CPU> per task
- User 2 wants <3 CPU> per task

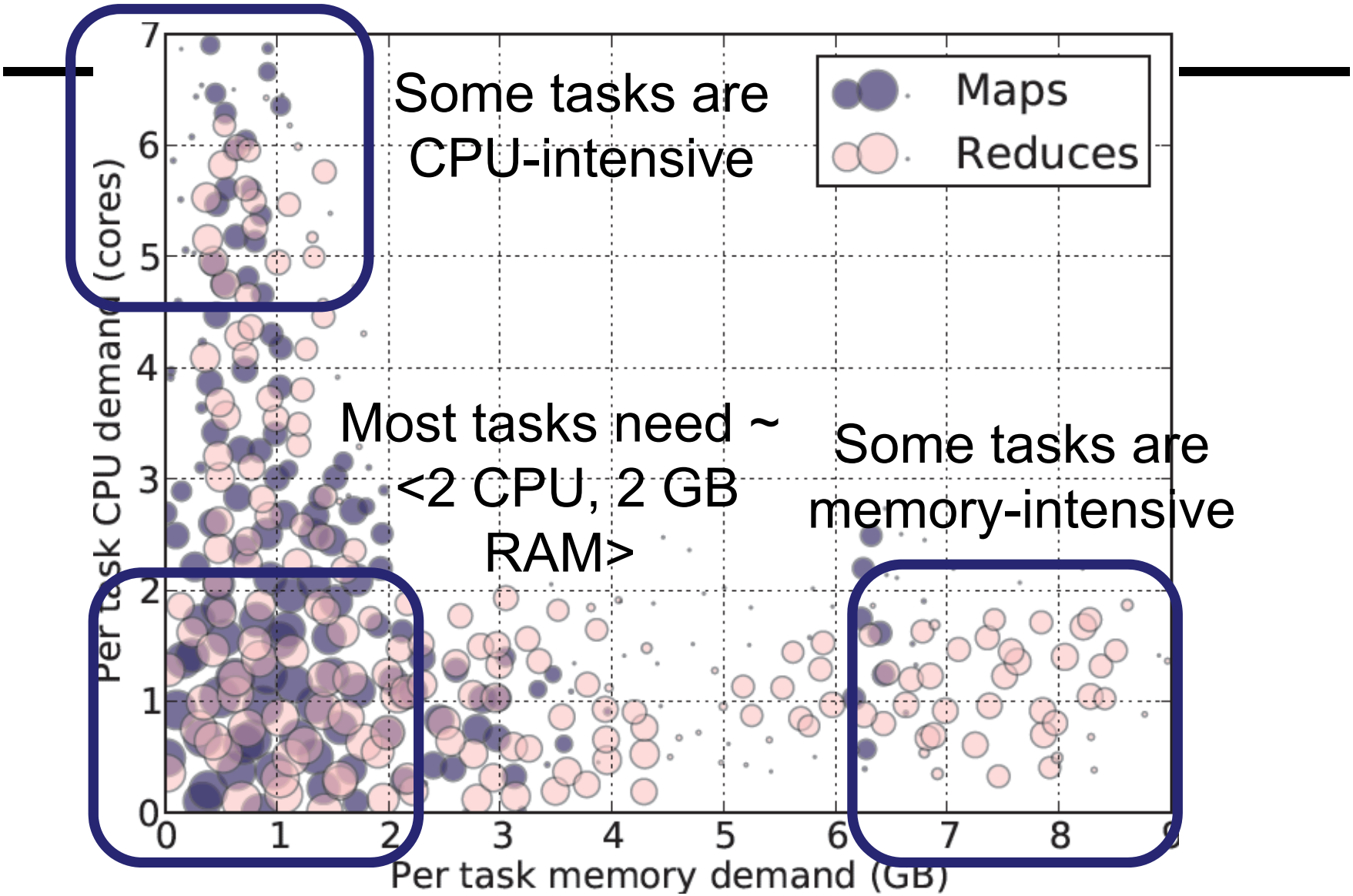


Multi-resource example

- 2 resources: CPUs & memory
- User 1 wants <1 CPU, 4 GB> per task
- User 2 wants <3 CPU, 1 GB> per task
- *What is a fair allocation?*



Heterogeneous Resource Demands



2000-node Hadoop Cluster at Facebook (Oct 2010)

Problem Definition

How to fairly share multiple resources when users/tasks have heterogeneous resource demands?

Model

- Users have *tasks* according to a *demand vector*
  - e.g., <2, 3, 1> user's tasks need 2 R_1, 3 R_2, 1 R_3
  - How to get the demand vectors is an interesting question

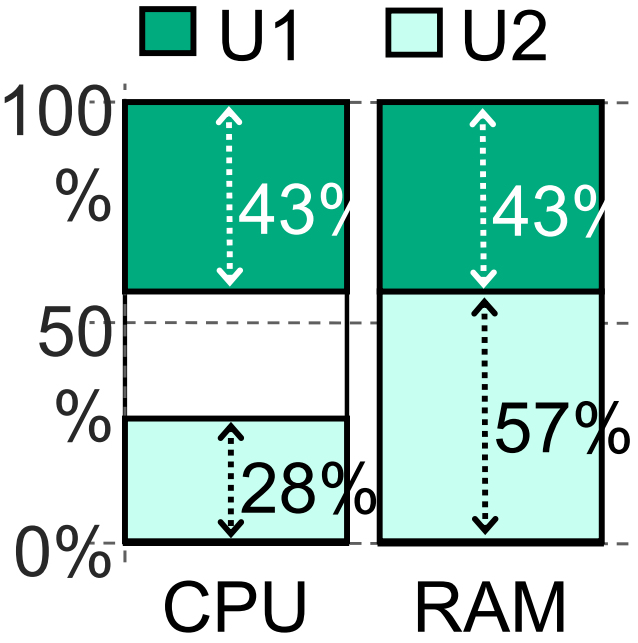
- Assume divisible resources

A Simple Solution: Asset Fairness

- *Asset Fairness*
  - Equalize each user's *sum of resource shares*

Problem:
User U1 has < 50% of both CPUs and RAM

Better off in a separate cluster with 50% of the resources



- Asset fairness yields
  - U_1: 15 tasks: 30 CPUs, 30 GB (∑=60)
  - U_2: 20 tasks: 20 CPUs, 40 GB (∑=60)

Share Guarantee

- Intuitively: “You shouldn’t be worse off than if you ran your own cluster with 1/*n* of the resources”
  - Otherwise, no incentive to share resources into a common pool
  - Each user should get at least 1/n of at least one resource (share guarantee)

Dominant Resource Fairness

- A user's *dominant resource* is the resource for which she has the biggest demand
  - Example:

    Total resources: <10 CPU, 4 GB>

    User 1's task requires: <2 CPU, 1 GB>

    Dominant resource is memory as 1/4 > 2/10 (1/5)

- A user's *dominant share* is the fraction of the dominant resource she is allocated

Dominant Resource Fairness (2)

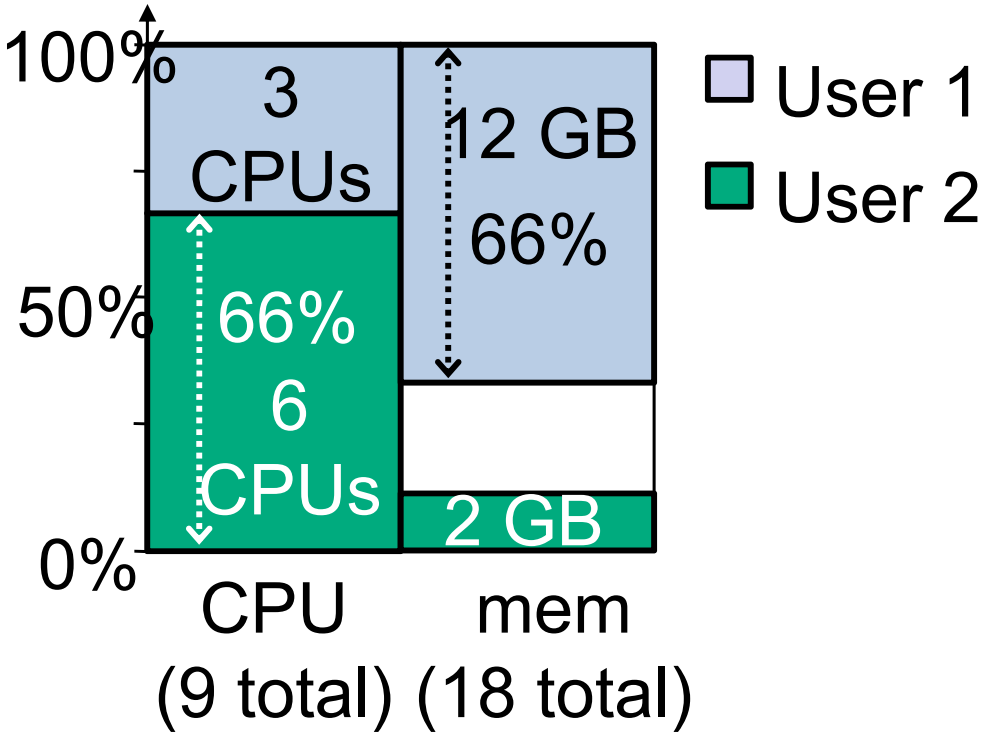
- Equalize the dominant share of the users

  Example:

  Total resources: <9 CPU, 18 GB>

  User 1 demand: <1 CPU, 4 GB> dominant res: mem

  User 2 demand: <3 CPU, 1 GB> dominant res: CPU



DRF is Fair and Much More

- DRF satisfies the share guarantee
- DRF is strategy-proof
- DRF allocations are envy-free

Cheating the Scheduler

- Some users will *game* the system to get more resources
- Real-life examples
  - A cloud provider had quotas on map and reduce slots
    Some users found out that the map quota was low
    - Users implemented maps in the reduce slots!
  - A search company provided dedicated machines to users that could ensure certain level of utilization (e.g. 80%)
    - Users used busy-loops to inflate utilization

Google's CPI² System

- Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, John Wilkes: CPI² -- CPU performance isolation for shared compute clusters. EuroSys 2013
  - Based on the application's cycles per instruction (CPI)
  - Observe the run-time performance of hundreds to thousands of tasks belonging to the same job, and learn to distinguish normal performance from outliers
  - Identify performance interference within a few minutes by detecting such outliers (victims)
  - Determine which antagonist applications are the likely cause with an online cross-correlation analysis
  - (if desired) Ameliorate the bad behavior by throttling or migrating the antagonists

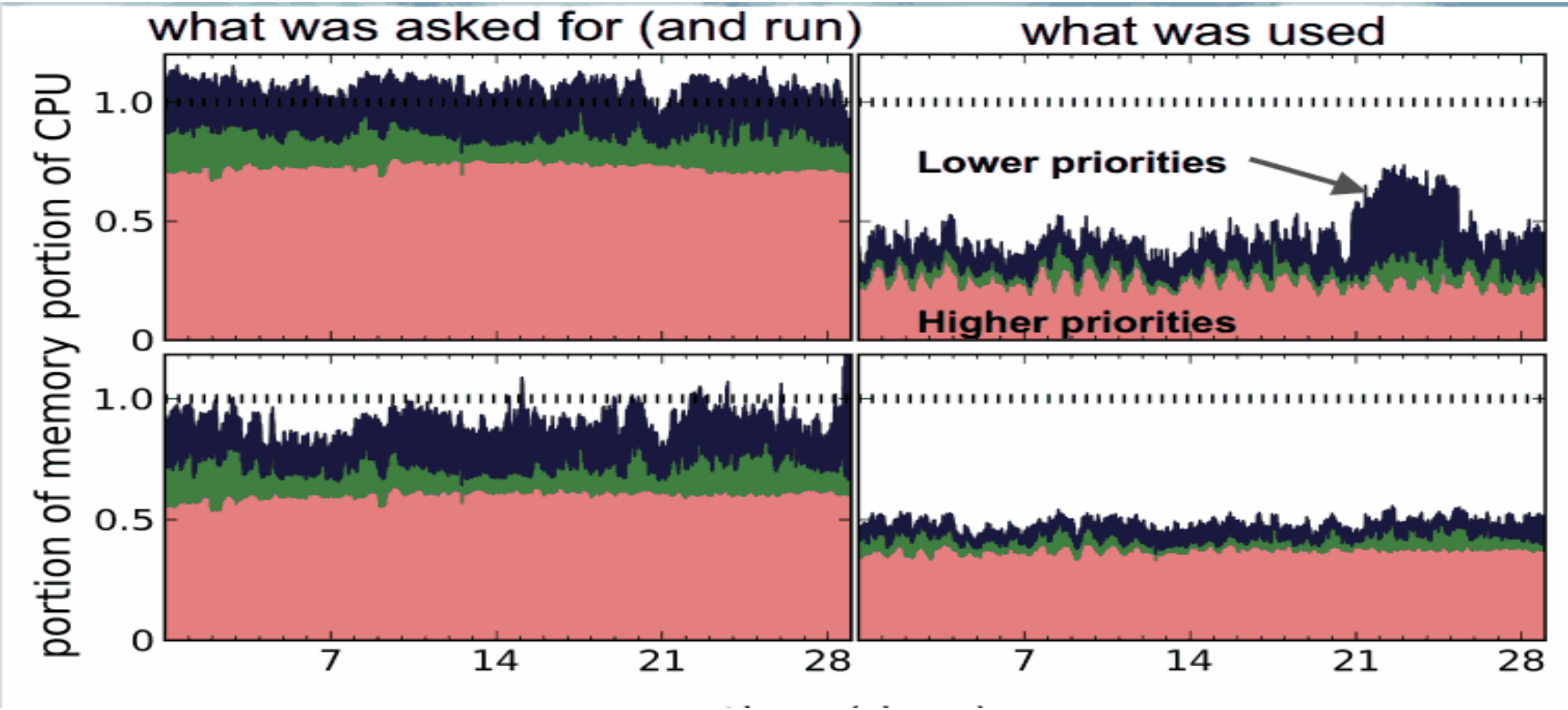
Seminar Outline

- Introduction
- Workload-level decisions in database systems
  - Physical design
  - Progress monitoring
  - Managing long running queries
- Performance prediction
- Progress Monitoring
- Inter workload interactions
- Outlook and Open Problems

Outlook

Large Scale Studies

Charles Reiss, Alexey Tumanov, Gregory Ganger, Randy Katz, and Michael Kozuch: Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. SOCC 2012



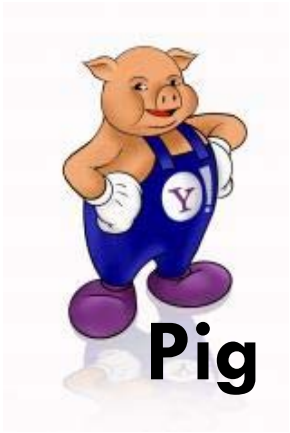
Large Scale Studies

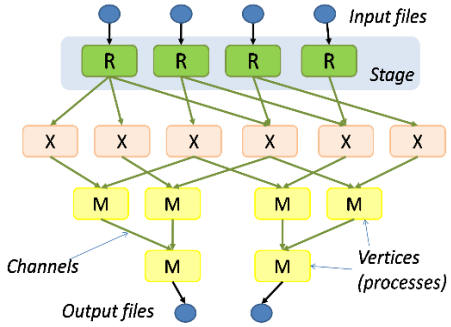
- Analysis of the Google trace breaks many common assumptions
  - Shows a big variation across task sizes
  - Shows that the scheduler is invoked very frequently
  - Shows that task resource demand vectors are not accurate
  - Shows a lot of machine heterogeneity

Resource Management Frameworks

- Rapid innovation in cluster computing frameworks

Google
Pregel





Dryad

CIEL

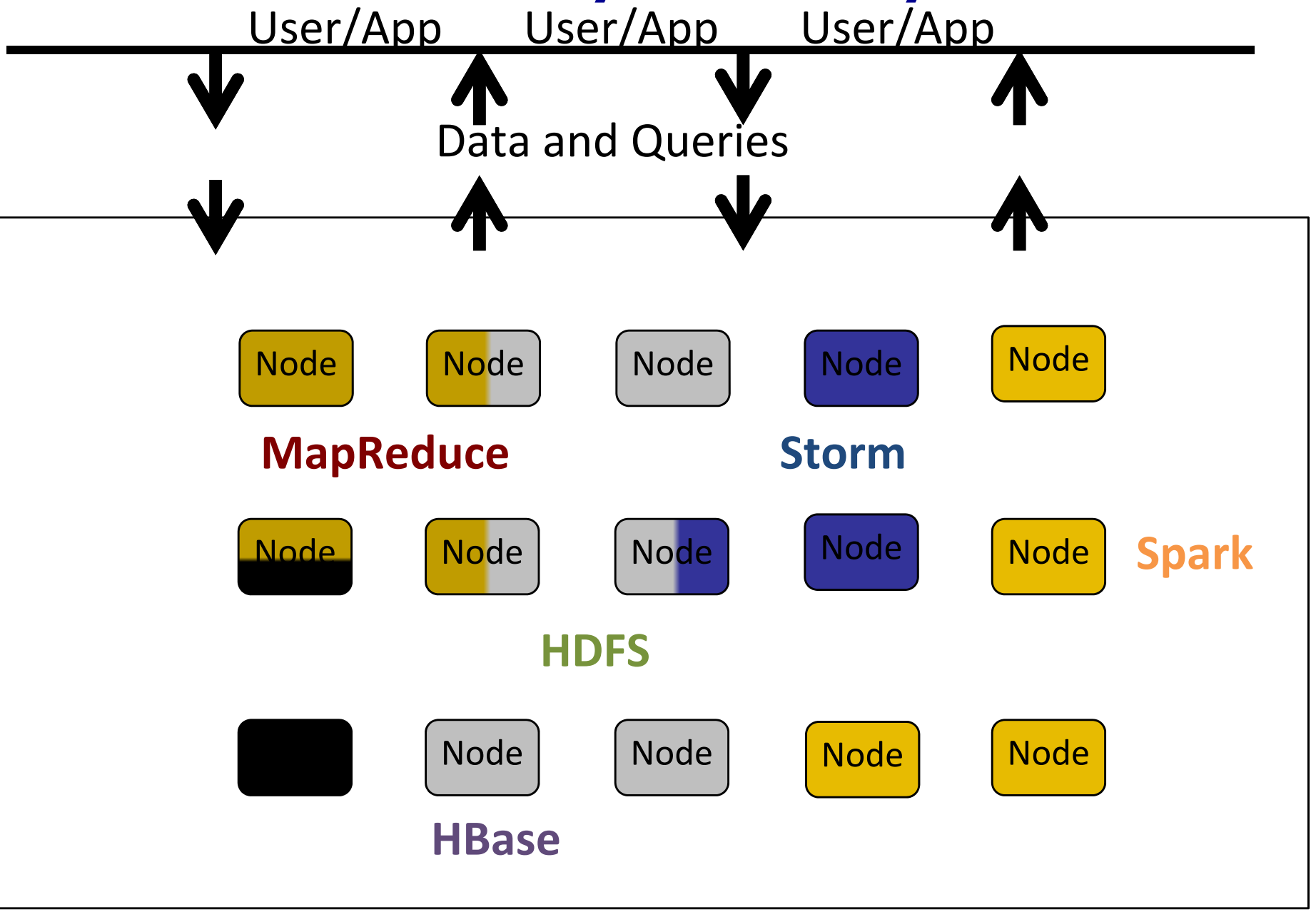
S4 distributed stream computing platform

Google
Percolator

Resource Management Frameworks

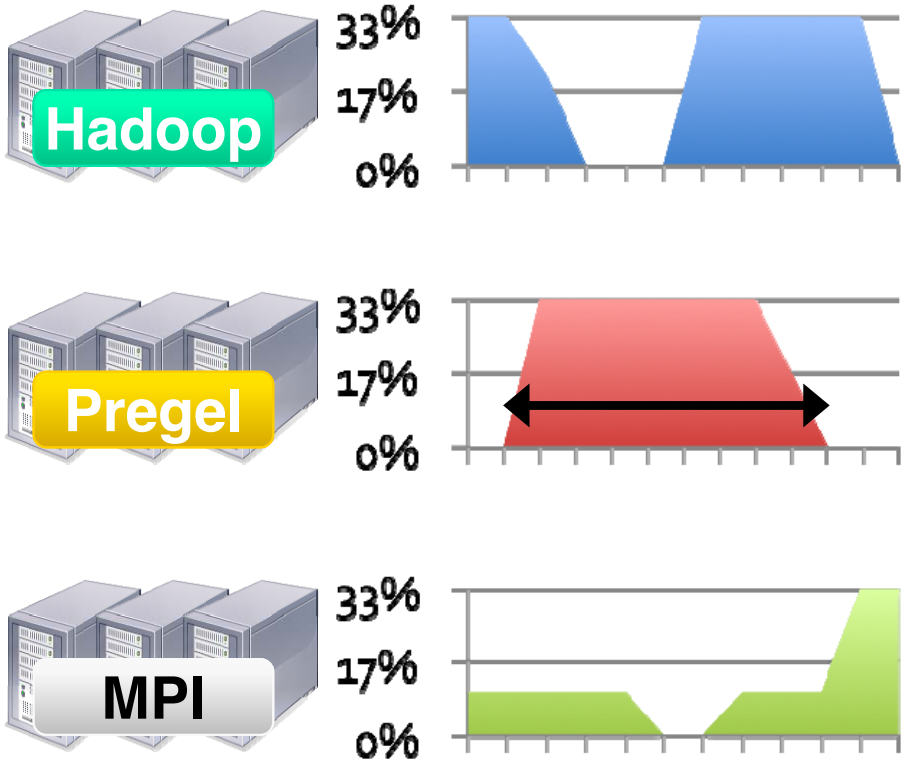
- Rapid innovation in cluster computing frameworks
- **No single framework optimal for all applications**
- Want to run multiple frameworks in a single cluster
  - … to *maximize utilization*
  - … to *share data* between frameworks

Multi-tenancy at Many Levels

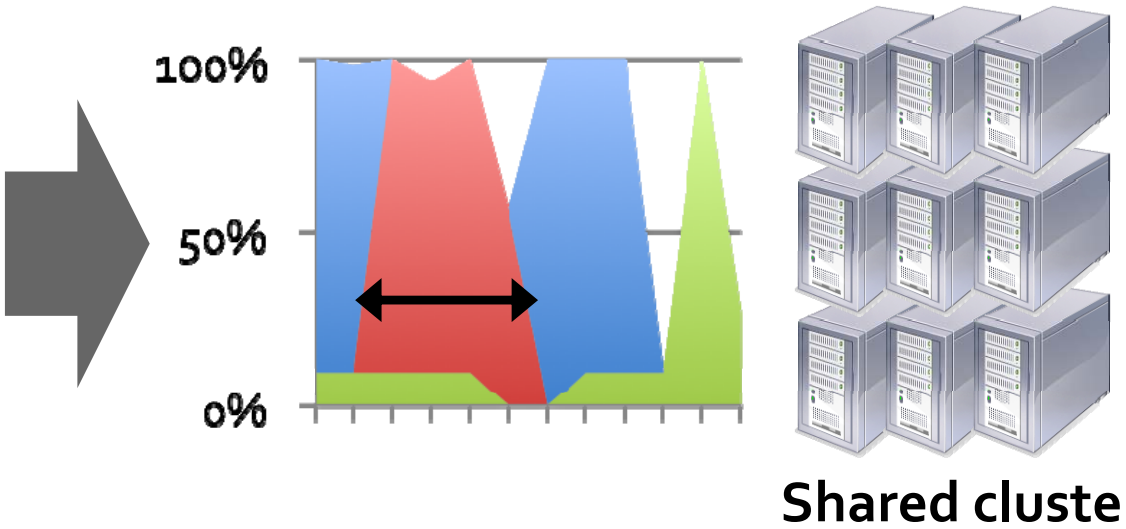


Where We Want to Go

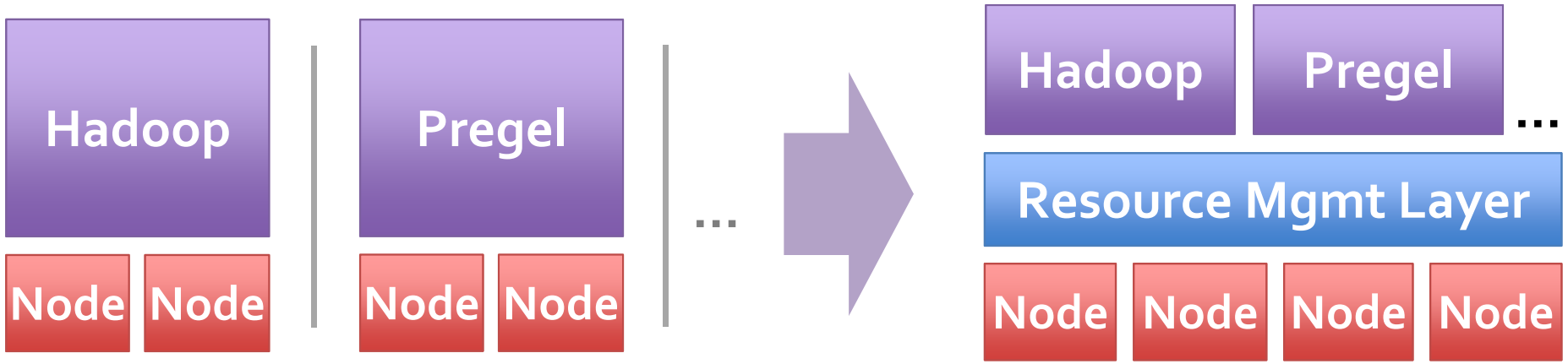
Today: static partitioning



Need: dynamic sharing



Resource Management Frameworks



- Mesos, YARN, Serengeti
- Also: run multiple instances of the *same* framework
  - Isolate production and experimental jobs
  - Run multiple versions of the framework concurrently
- Lots of challenges!

Workload Management Benchmarks are Needed

- Tim Kiefer, Benjamin Schlegel, and Wolfgang Lehner: MulTe: A Multi-Tenancy Database Benchmark Framework. LNCS Vol 7755, 2013
- Running single-tenant benchmarks in a multi-tenant setting
- Uses existing TPC-H benchmark queries and data
  - Measures scalability, isolation, and fairness
  - User defines each tenant's workload and sets up a scenario.
- Defining Tenant example:
  - Type: TPC-H, Size: 500 MB, Query: Query 1, MeanSleepTime: 0, Parallel Users: 5, Activity: 300, ActivityConstraint: seconds
- Workload Driver runs tenants' workloads and gives results.
- Their Workload Driver is an extension of the TPoX database benchmark's workload driver

Challenges (1/2)

- Integrating the notion of a workload in traditional systems
  - Query optimization
  - Scheduling
- Managing workload interactions
  - Better workload isolation
  - Inducing more positive interactions
- Multi-tenancy and cloud
  - More workloads to interact with each other
  - Opportunities for shared optimizations
  - Heterogeneous infrastructure
  - Elastic infrastructure
  - Scale

Challenges (2/2)

- Better performance modeling
  - Especially for MapReduce
- Rich yet simple definition of SLOs
  - Dollar cost
  - Failure
  - Fuzzy penalties
  - Scale

References (1/4)

- Mumtaz Ahmad, Ashraf Aboulnaga, Shivnath Babu, Kamesh Munagala. “Interaction-Aware Scheduling of Report Generation Workloads.” VLDBJ 2011.
- Mumtaz Ahmad, Songyun Duan, Ashraf Aboulnaga, Shivnath Babu. “Predicting Completion Times of Batch Query Workloads Using Interaction-aware Models and Simulation.” EDBT 2011.
- Mert Akdere, Ugur Cetintemel, Matteo Riondato, Eli Upfal, Stanley B. Zdonik. “Learning-based Query Performance Modeling and Prediction.” ICDE 2012.
- Kurt P. Brown, Manish Mehta, Michael J. Carey, Miron Livny. “Towards Automated Performance Tuning for Complex Workloads.” VLDB 1994.
- Surajit Chaudhuri, Vivek Narasayya. “Self-Tuning Database Systems: A Decade of Progress.” VLDB 2007.
- Whei-Jen Chen, Bill Comeau, Tomoko Ichikawa, S Sadish Kumar, Marcia Miskimen, H T Morgan, Larry Pay, Tapio Väättänen. “DB2 Workload Manager for Linux, UNIX, and Windows.” IBM Redbook 2008.
- Yanpei Chen, Sara Alspaugh, Randy Katz. “Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads.” VLDB 2012.

References (2/4)

- Jennie Duggan, Ugur Cetintemel, Olga Papaemmanouil, Eli Upfal. “Performance Prediction for Concurrent Database Workloads.” SIGMOD 2011.
- Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox, Michael Jordan, David Patterson. “Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning.” ICDE 2009.
- A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica. “Dominant Resource Fairness: Fair Allocation of Multiple Resources Types.” NSDI 2011.
- Chetan Gupta, Abhay Mehta, Song Wang, Umeshwar Dayal. “Fair, Effective, Efficient and Differentiated Scheduling in an Enterprise Data Warehouse.” EDBT 2009.
- Herodotos Herodotou, Shivnath Babu. “Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs.” VLDB 2011.
- B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, I. Stoica. “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.” NSDI 2011.
- Stefan Krompass, Harumi Kuno, Janet L. Wiener, Kevin Wilkinson, Umeshwar Dayal, Alfons Kemper. “Managing Long-Running Queries.” EDBT 2009.

References (3/4)

- Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, Andrew Goldberg. “Quincy: fair scheduling for distributed computing clusters.” SOSP 2009.
- Arnd Christian Konig, Bolin Ding, Surajit Chaudhuri, Vivek Narasayya. “A Statistical Approach Towards Robust Progress Estimation.” VLDB 2012.
- Jiexing Li, Arnd Christian Konig, Vivek Narasayya, Surajit Chaudhuri. “Robust Estimation of Resource Consumption for SQL Queries using Statistical Techniques.” VLDB 2012.
- Jiexing Li, Rimma V. Nehme, Jeffrey Naughton. “GSLPI: a Cost-based Query Progress Indicator.” ICDE 2012.
- Gang Luo, Jeffrey F. Naughton, Philip S. Yu. “Multi-query SQL Progress Indicators.” EDBT 2006.
- Kristi Morton, Magdalena Balazinska, Dan Grossman. “ParaTimer: A Progress Indicator for MapReduce DAGs.” SIGMOD 2010.
- Baoning Niu, Patrick Martin, Wendy Powley, Paul Bird, Randy Horman. “Adapting Mixed Workloads to Meet SLOs in Autonomic DBMSs.” SMDB 2007.
- HweeHwa Pang, Michael J. Carey, Miron Livny. “Multiclass Query Scheduling in Real-Time Database Systems.” IEEE TKDE 1995.

References (4/4)

- Sujay S. Parekh, Kevin Rose, Joseph L. Hellerstein, Sam Lightstone, Matthew Huras, Victor Chang. “Managing the Performance Impact of Administrative Utilities.” DSOM 2003.
- Wentao Wu, Yun Chi, Shenghuo Zhu, Junichi Tatemura, Hakan Hacigümüs, Jeffrey F. Naughton. “Predicting Query Execution Time: Are Optimizer Cost Models Really Unusable?” ICDE 2013
- M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica. “Job Scheduling for Multi-User MapReduce Clusters.” UC Berkeley Technical Report UCB/EECS-2009-55, April 2009.

Acknowledgements

- We would like to thank the authors of the referenced papers for making their slides available on the web. We have borrowed generously from such slides in order to create this presentation.