EECS 262a
Advanced Topics in Computer Systems
Lecture 16

Comparison of Parallel DB, CS, MR and Jockey
March 16th, 2016

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs262

Today's Papers

- A Comparison of Approaches to Large-Scale Data Analysis
  Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker. Appears in *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 2009
- Jockey: Guaranteed Job Latency in Data Parallel Clusters
  Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Appears in *Proceedings of the European Professional Society on Computer Systems* (EuroSys), 2012

- Thoughts?

Two Approaches to Large-Scale Data Analysis

- **"Shared nothing"**
- **MapReduce**
  - **Distributed file system**
  - **Map, Split, Copy, Reduce**
  - **MR scheduler**
- **Parallel DBMS**
  - **Standard relational tables, (physical location transparent)**
  - **Data are partitioned over cluster nodes**
  - **SQL**
  - **Join processing: T1 joins T2**
    - **If T2 is small, copy T2 to all the machines**
    - **If T2 is large, then hash partition T1 and T2 and send partitions to different machines (this is similar to the split-copy in MapReduce)**
  - **Query Optimization**
  - **Intermediate tables *not* materialized by default**

Architectural Differences

	Parallel DBMS	MapReduce
Schema Support	O	X
Indexing	O	X
Programming Model	Stating what you want (SQL)	Presenting an algorithm (C/C++, Java, …)
Optimization	O	X
Flexibility	Spotty UDF Support	Good
Fault Tolerance	Not as Good	Good
Node Scalability	<100	>10,000

Schema Support

- MapReduce
  - Flexible, programmers write code to interpret input data
  - Good for single application scenario
  - Bad if data are shared by multiple applications. Must address data syntax, consistency, etc.
- Parallel DBMS
  - Relational schema required
  - Good if data are shared by multiple applications

Programming Model & Flexibility

- MapReduce
  - Low level: "We argue that MR programming is somewhat analogous to Codasyl programming…"
  - "Anecdotal evidence from the MR community suggests that there is widespread sharing of MR code fragments to do common tasks, such as joining data sets."
  - very flexible
- Parallel DBMS
  - SQL
  - user-defined functions, stored procedures, user-defined aggregates

Indexing

- MapReduce
  - No native index support
  - Programmers can implement their own index support in Map/Reduce code
  - But hard to share the customized indexes in multiple applications
- Parallel DBMS
  - Hash/b-tree indexes well supported

Execution Strategy & Fault Tolerance

- MapReduce
  - Intermediate results are saved to local files
  - If a node fails, run the node-task again on another node
  - At a mapper machine, when multiple reducers are reading multiple local files, there could be large numbers of disk seeks, leading to poor performance.
- Parallel DBMS
  - Intermediate results are pushed across network
  - If a node fails, must re-run the entire query

Avoiding Data Transfers

- MapReduce
  - Schedule Map close to data
  - But other than this, programmers must avoid data transfers themselves
- Parallel DBMS
  - A lot of optimizations
  - Such as determine where to perform filtering

Node Scalability

- MapReduce
  - 10,000's of commodity nodes
  - 10's of Petabytes of data
- Parallel DBMS
  - <100 expensive nodes
  - Petabytes of data

Performance Benchmarks

- Benchmark Environment
- Original MR task (Grep)
- Analytical Tasks
  - Selection
  - Aggregation
  - Join
  - User-defined-function (UDF) aggregation

Node Configuration

- 100-node cluster
  - Each node: 2.40GHz Intel Core 2 Duo, 64-bit red hat enterprise Linux 5 (kernel 2.6.18) w/ 4Gb RAM and two 250GB SATA HDDs.
- Nodes interconnected with Cisco Catalyst 3750E 1Gb/s switches
  - Internal switching fabric has 128Gbps
  - 50 nodes per switch
- Multiple switches interconnected via 64Gbps Cisco StackWise ring
  - The ring is only used for cross-switch communications.

Tested Systems

- Hadoop (0.19.0 on Java 1.6.0)
  - HDFS data block size: 256MB
  - JVMs use 3.5GB heap size per node
  - "Rack awareness" enabled for data locality
  - Three replicas w/o compression: Compression or fewer replicas in HDFS does not improve performance
- DBMS-X (a parallel SQL DBMS from a major vendor)
  - Row store
  - 4GB shared memory for buffer pool and temp space per node
  - Compressed table (compression often reduces time by 50%)
- Vertica
  - Column store
  - 256MB buffer size per node
  - Compressed columns by default

Benchmark Execution

- Data loading time:
  - Actual loading of the data
  - Additional operations after the loading, such as compressing or building indexes
- Execution time
  - DBMS-X and vertica:
    - » Final results are piped from a shell command into a file
  - Hadoop:
    - » Final results are stored in HDFS
    - » An *additional* Reduce job step to combine the multiple files into a single file

Performance Benchmarks

- Benchmark Environment
- Original MR task (Grep)
- Analytical Tasks
  - Selection
  - Aggregation
  - Join
  - User-defined-function (UDF) aggregation

Task Description

- From MapReduce paper
  - Input data set: 100-byte records
  - Look for a three-character pattern
  - One match per 10,000 records
- Varying the number of nodes
  - Fix the size of data per node (535MB/node)
  - Fix the total data size (1TB)

Data Loading

- **Hadoop:**
  - Copy text files into HDFS in parallel
- **DBMS-X:**
  - Load SQL command executed in parallel: it performs hash partitioning and distributes records to multiple machines
  - Reorganize data on each node: compress data, build index, perform other housekeeping
    - » This happens in parallel
- **Vertica:**
  - Copy command to load data in parallel
  - Data is re-distributed, then compressed

Data Loading Times

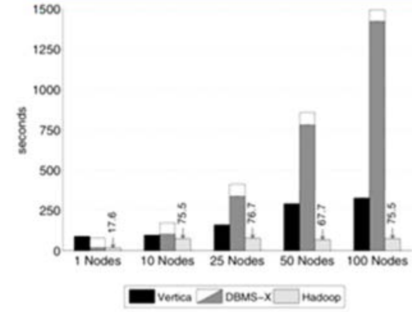


Figure 1: Load Times – Grep Task Data Set (535MB/node)

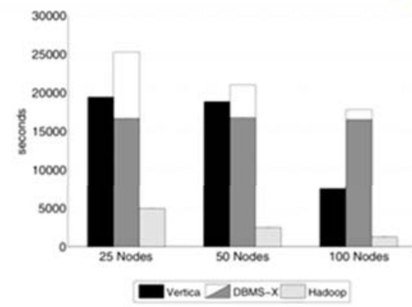


Figure 2: Load Times – Grep Task Data Set (1TB/cluster)

- **DBMS-X: grey is loading, white is re-organization after loading**
  - Loading is actually sequential despite parallel load commands
- **Hadoop does better because it only copies the data to three HDFS replicas**

Execution

- **SQL:**
  - SELECT * FROM data WHERE field LIKE "%XYZ%"
  - Full table scan
- **MapReduce:**
  - Map: pattern search
  - No reduce
  - An additional Reduce job to combine the output into a single file

Execution time

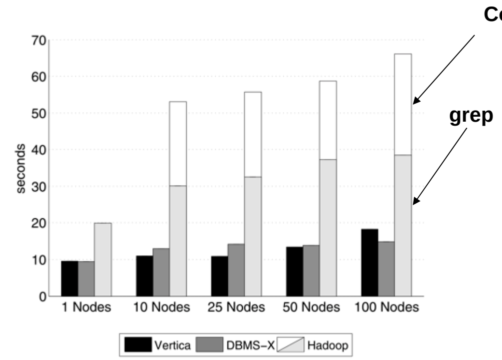


Figure 4: Grep Task Results – 535MB/node Data Set

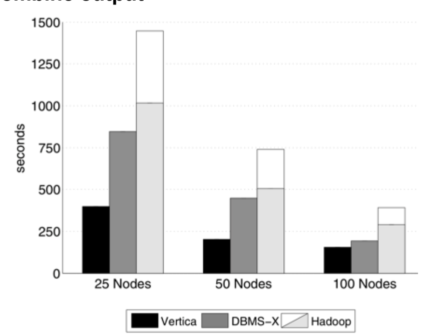


Figure 5: Grep Task Results – 1TB/cluster Data Set

- **Hadoop's large start-up cost shows up in Figure 4, when data per node is small**
- **Vertica's good data compression**

Performance Benchmarks

- Benchmark Environment
- Original MR task (Grep)
- Analytical Tasks
  - Selection
  - Aggregation
  - Join
  - User-defined-function (UDF) aggregation

Input Data

- Input #1: random HTML documents
  - Inside an html doc, links are generated with Zipfian distribution
  - 600,000 unique html docs with unique urls per node
- Input #2: 155 million UserVisits records
  - 20GB/node
- Input #3: 18 million Ranking records
  - 1GB/node

Selection Task

- Find the pageURLs in the rankings table (1GB/node) with a pageRank > threshold
  - 36,000 records per data file (very selective)
- SQL:

  SELECT pageURL, pageRank
  FROM Rankings WHERE pageRank > X;

- MR: single Map, no Reduce

Selection Task

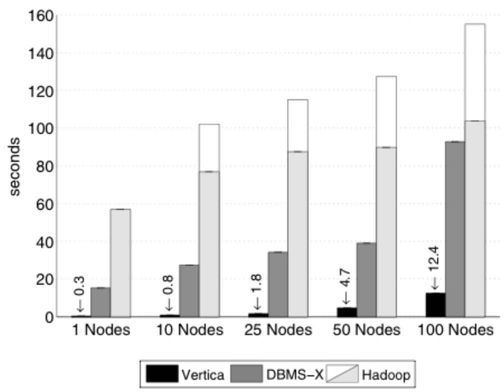


Figure 6: Selection Task Results

- Hadoop's start-up cost; DBMS uses index; vertica's reliable message layer becomes bottleneck

Aggregation Task

- **Calculate the total adRevenue generated for each sourceIP in the UserVisits table (20GB/node), grouped by the sourceIP column.**
  - **Nodes must exchange info for computing groupby**
  - **Generate 53 MB data regardless of number of nodes**
- **SQL:**

  SELECT sourceIP, SUM(adRevenue)
  FROM UserVisits GROUP BY sourceIP;
- **MR:**
  - **Map: outputs (sourceIP, adRevenue)**
  - **Reduce: compute sum per sourceIP**
  - **"Combine" is used**

Aggregation Task

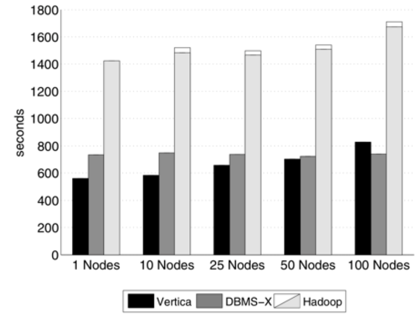


Figure 7: Aggregation Task Results (2.5 million Groups)

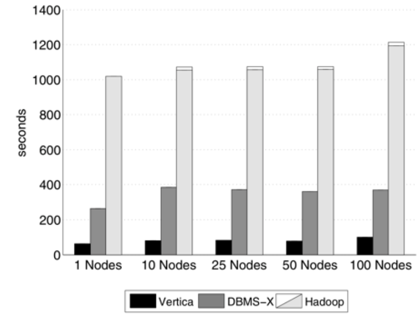


Figure 8: Aggregation Task Results (2,000 Groups)

- **DBMS: Local group-by, then the coordinator performs the global group-by; performance dominated by data transfer.**

Join Task

- **Find the sourceIP that generated the most revenue within Jan 15-22, 2000, then calculate the average pageRank of all the pages visited by the sourceIP during this interval**
- **SQL:**

```
SELECT INTO Temp sourceIP,
              AVG(pageRank) as avgPageRank,
             SUM(adRevenue) as totalRevenue
FROM   Rankings AS R, UserVisits AS UV
WHERE R.pageURL = UV.destURL
              AND UV.visitDate BETWEEN Date('2000-01-15')
              AND Date('2000-01-22')
GROUP BY UV.sourceIP;

SELECT sourceIP, totalRevenue, avgPageRank
FROM Temp
ORDER BY totalRevenue DESC LIMIT 1;
```


Map Reduce

- **Phase 1: filter UserVisits that are outside the desired date range, joins the qualifying records with records from the Ranking file**
- **Phase 2: compute total adRevenue and average pageRank per sourceIP**
- **Phase 3: produce the largest record**

Join Task

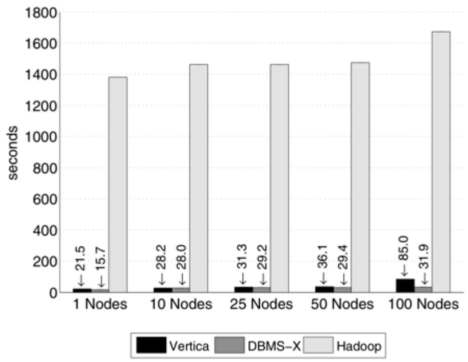


Figure 9: Join Task Results

- **DBMS can use index, both relations are partitioned on the join key; MR has to read all data**
- **MR phase 1 takes an average 1434.7 seconds**
  - **600 seconds of raw I/O to read the table; 300 seconds to split, parse, deserialize; Thus CPU overhead is the limiting factor**

UDF Aggregation Task

- **Compute inlink count per document**
- **SQL:**

```
SELECT INTO Temp F(contents) FROM Documents;
SELECT url, SUM(value) FROM Temp GROUP BY url;
```

Need a user-defined-function to parse HTML docs (C pgm using POSIX regex lib)

Both DBMS's do not support UDF very well, requiring separate program using local disk and bulk loading of the DBMS – *why was MR always forced to use Reduce to combine results?*

- **MR:**
  - **A standard MR program**

UDF Aggregation

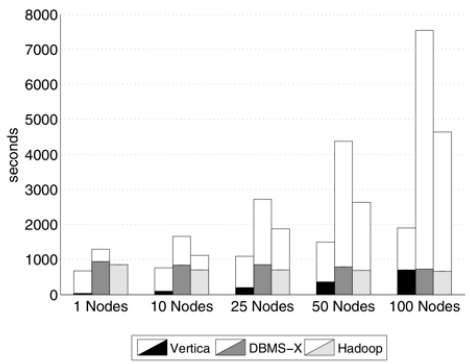


Figure 10: UDF Aggregation Task Results

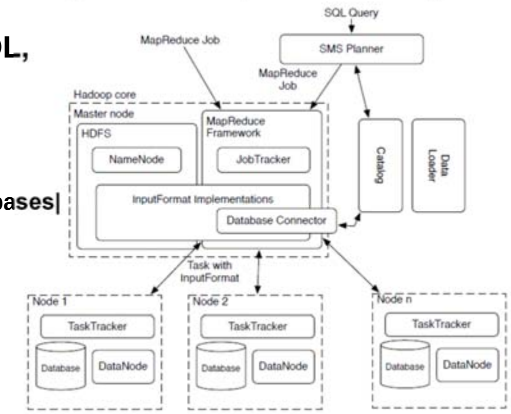
- **DBMS: lower – UDF time; upper – other query time**
- **Hadoop: lower – query time; upper: combine all results into one**

Discussion

- **Throughput experiments?**
- **Parallel DBMSs are much more challenging than Hadoop to install and configure properly – DBMSs require professional DBAs to configure/tune**
- **Alternatives: Shark (Hive on Spark)**
  - **Eliminates Hadoop task start-up cost and answers queries with sub-second latencies**
    - » **100 node system: 10 second till the first task starts, 25 seconds till all nodes run tasks**
  - **Columnar memory store (multiple orders of magnitude faster than disk**
- **Compression: does not help in Hadoop?**
  - **An artifact of Hadoop's Java-based implementation?**
- **Execution strategy (DBMS), failure model (Hadoop), ease of use (H/D)**
- **Other alternatives? Apache Hive, Impala (Cloudera) , HadoopDB (Hadapt), …**

Alternative: HadoopDB?

- **The Basic Idea (An Architectural Hybrid of MR & DBMS)**
  - **To use MR as the communication layer above multiple nodes running single-node DBMS instances**
- **Queries expressed in SQL, translated into MR by extending existing tools**
  - **As much work as possible is pushed into the higher performing single node databases|**
- **How many of complaints from Comparison paper still apply here?**
- **Hadapt startup commercializing**

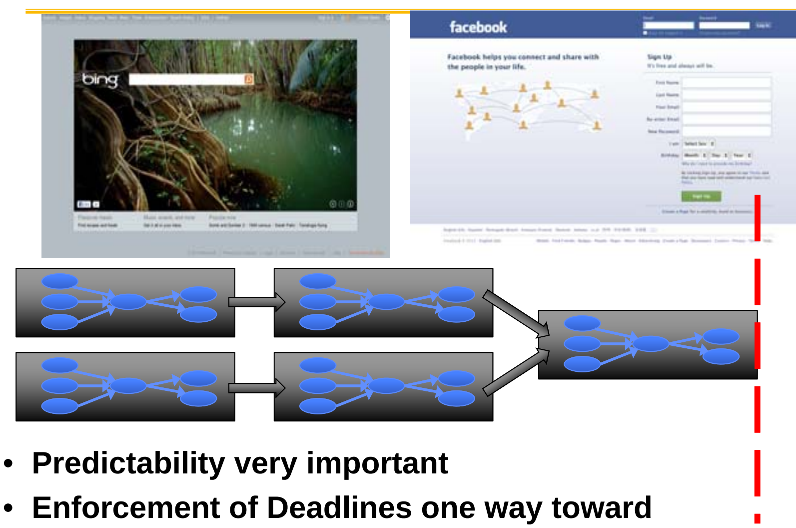


Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?

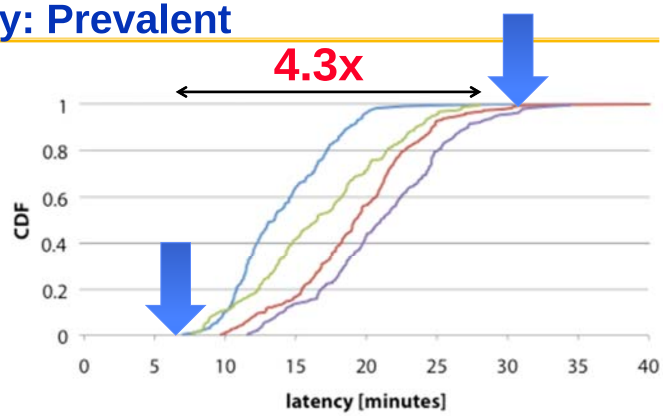
BREAK

Domain for Jockey: Large cluster jobs



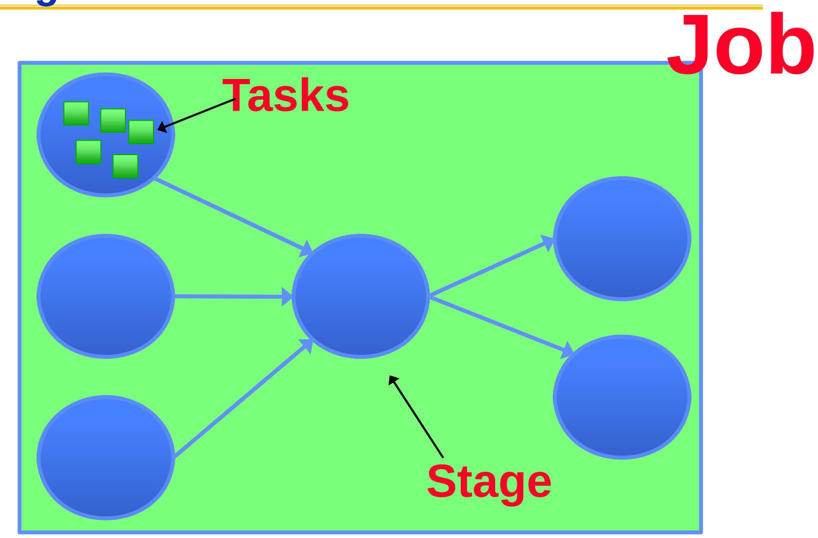
- **Predictability very important**
- **Enforcement of Deadlines one way toward predictability**

Variable Execution Latency: Prevalent

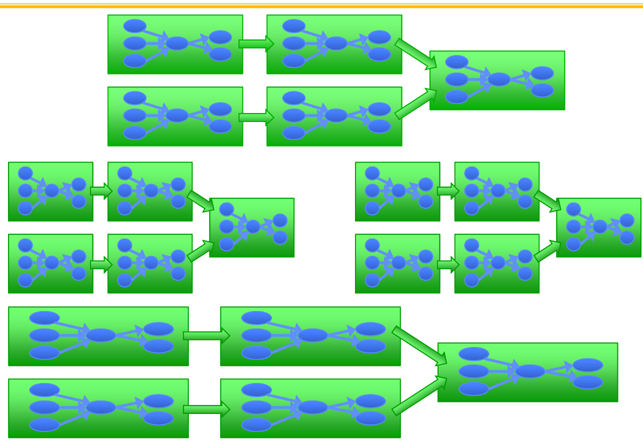


- **Even for job with narrowest latency profile**
  - Over 4.3X variation in latency
- **Reasons for latency variation:**
  - Pipeline complexity
  - Noisy execution environment
  - Excess resources

Job Model: Graph of Interconnected Stages

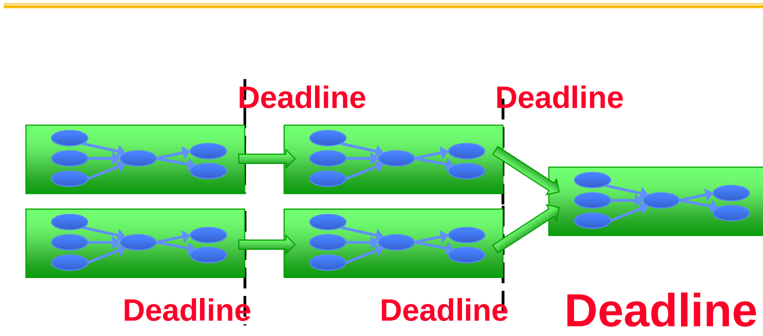


Dryad's Dag Workflow



- **Many simultaneous job pipelines executing at once**
- **Some on behalf of Microsoft, others on behalf of customers**

Compound Workflow



- **Dependencies mean that deadlines on complete pipeline create deadlines on constituent jobs**
- **Median job's output used by 10 additional jobs**

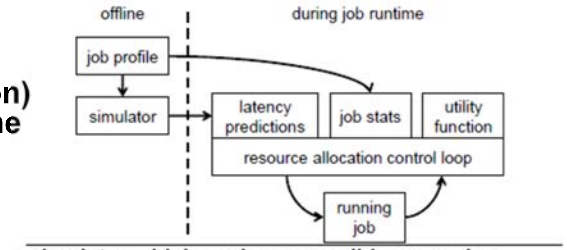
Best way to express performance targets

- **Priorities?** **Not expressive enough**
- **Weights?** **Difficult for users to set**
- **Utility curves?** **Capture deadline & penalty**

- **Jockey's goal:**
  Maximize utility while minimizing resources by dynamically adjusting the allocation

Application Modeling

C(progress, allocation) → remaining run time



- **Techniques:**
  - **Job simulator:**
    - » **Input from profiling to a simulator which explores possible scenarios**
    - » **Compute**
  - **Amdahl's Law**
    - » **Time = S + P/N**
    - » **Estimate S and P from standpoint of current stage**
- **Progress metric? Many explored**
  - ***totalworkWithQ:*** **Total time completed tasks spent enqueued or executing**
- **Optimization: Minimum allocation that maximizes utility**
- **Control loop design: slack (1.2), hysteresis, dead zone (D)**

Ex: Completion (1%), Deadline(50 min)

	10 nodes	20 nodes	30 nodes
1% complete	60 minutes	40 minutes	25 minutes
2% complete	59 minutes	39 minutes	24 minutes
3% complete	58 minutes	37 minutes	22 minutes
4% complete	56 minutes	36 minutes	21 minutes
5% complete	54 minutes	34 minutes	20 minutes

JOCKEY – CONTROL LOOP

Ex: Completion (3%), Deadline(50 min)

	10 nodes	20 nodes	30 nodes
1% complete	60 minutes	40 minutes	25 minutes
2% complete	59 minutes	39 minutes	24 minutes
3% complete	58 minutes	37 minutes	22 minutes
4% complete	56 minutes	36 minutes	21 minutes
5% complete	54 minutes	34 minutes	20 minutes

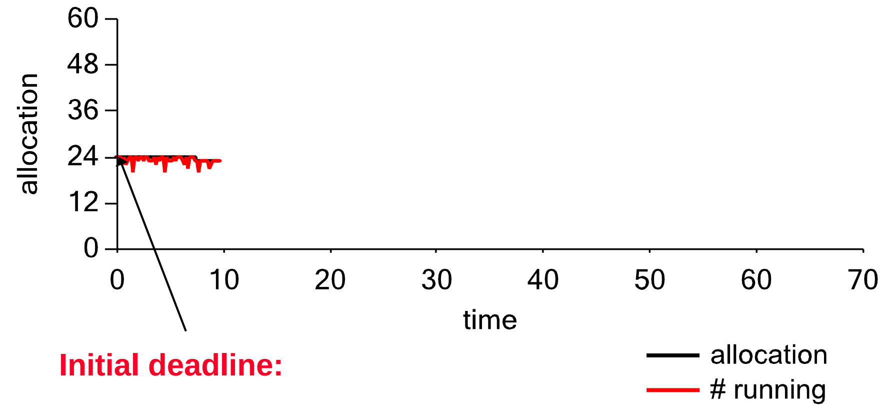
JOCKEY – CONTROL LOOP

Ex: Completion (5%), Deadline(30 min)

	10 nodes	20 nodes	30 nodes
1% complete	60 minutes	40 minutes	25 minutes
2% complete	59 minutes	39 minutes	24 minutes
3% complete	58 minutes	37 minutes	22 minutes
4% complete	56 minutes	36 minutes	21 minutes
5% complete	54 minutes	34 minutes	20 minutes

JOCKEY – CONTROL LOOP

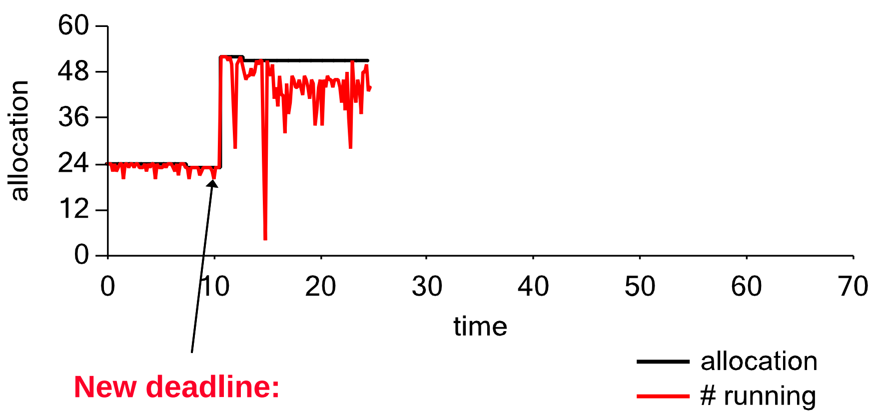
Jockey in Action



Initial deadline:

140 minutes

Jockey in Action

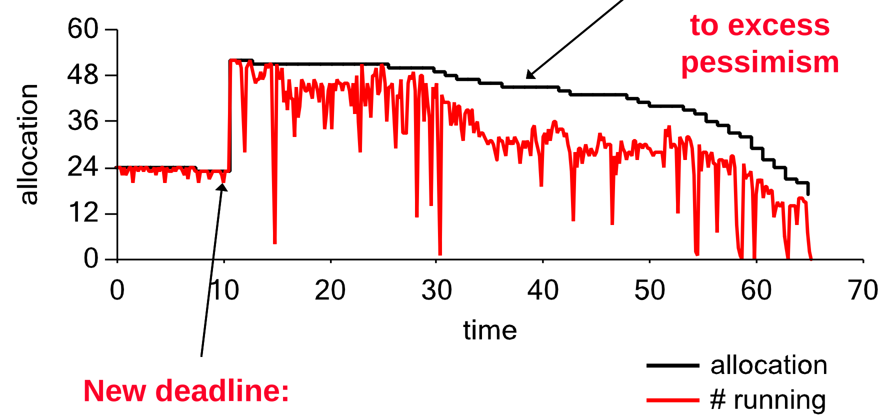


New deadline:

70 minutes

Jockey in Action

Release resources due to excess pessimism

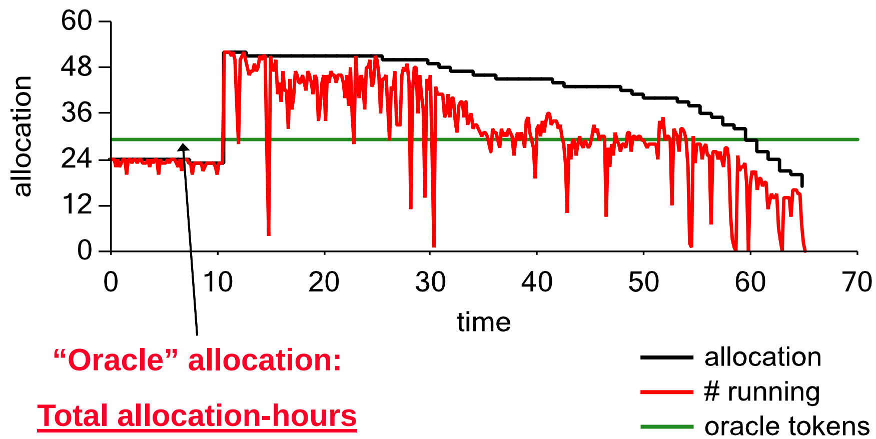


New deadline:

70 minutes

48

Jockey in Action



"Oracle" allocation:

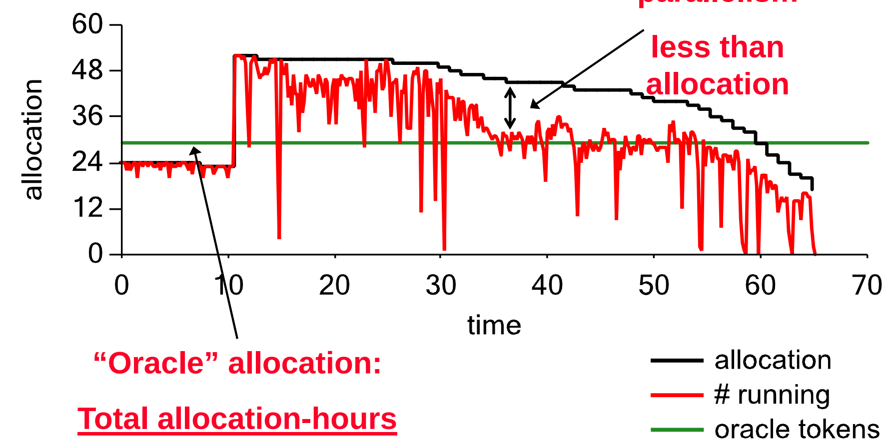
Total allocation-hours

Deadline

Jockey in Action

Available parallelism

less than allocation



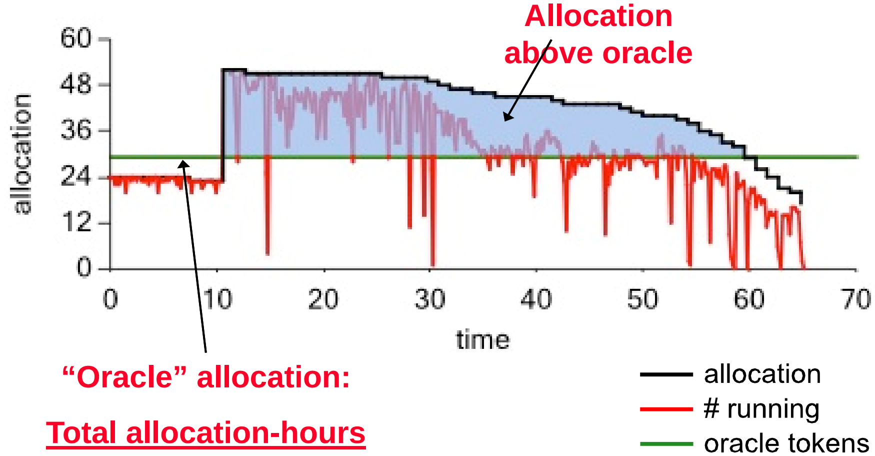
"Oracle" allocation:

Total allocation-hours

Deadline

Jockey in Action

Allocation above oracle



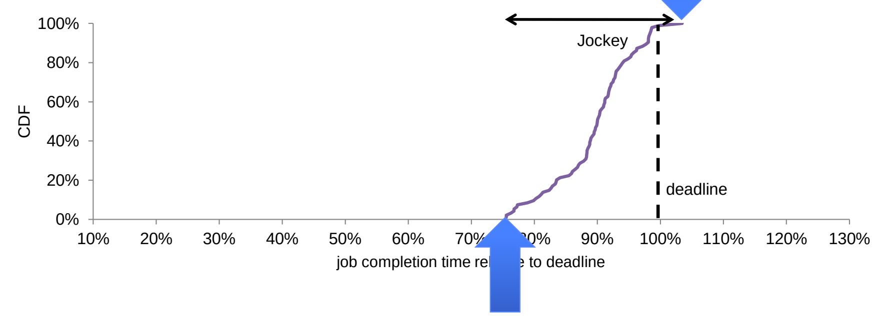
"Oracle" allocation:

Total allocation-hours

Deadline

Evaluation

1.4x



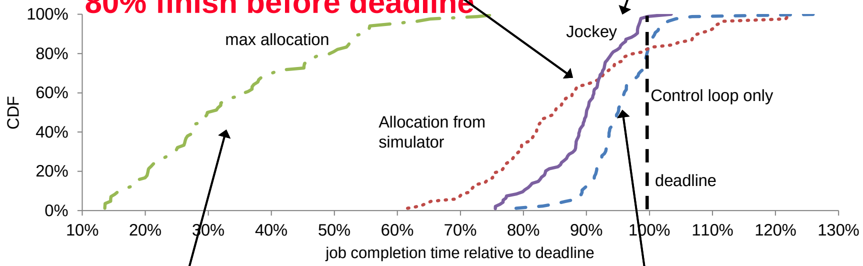
Jobs which met the SLO

Evaluation

Missed 1 of 94 deadlines

Simulator made good predictions:

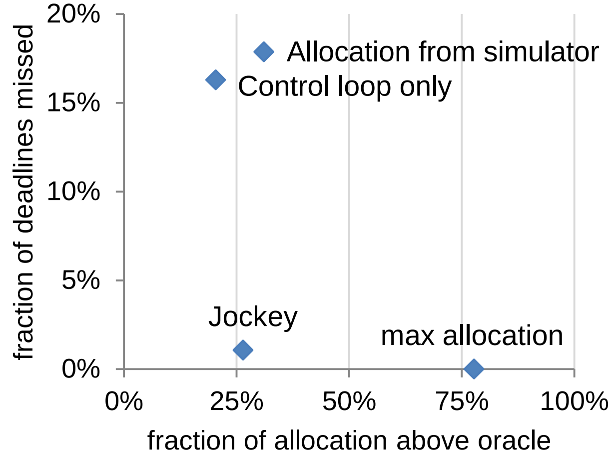
80% finish before deadline



Allocated too many resources

Control loop is stable and successful

Evaluation



Is this a good paper?

- What were the authors' goals?
- What about the evaluation/metrics?
- Did they convince you that this was a good system/approach?
- Were there any red-flags?
- What mistakes did they make?
- Does the system/approach meet the "Test of Time" challenge?
- How would you review this paper today?