Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources

Minos N. Garofalakis
Yannis E. Ioannidis
University of Wisconsin – Madison

Outline

- Introduction and Motivation
- Problem Formulation and Query Execution Model
- Heuristic Query Scheduling Algorithms
- Experimental Results
- Implications for Parallel Query Optimization
- Ongoing and Future Work

Introduction and Motivation

The Problem

- Architecture: Shared-Nothing or Hierarchical
- *Static* parallelization of set of complex query plans, based on size & cost estimates
- Objective: minimize query response time

Why?

- Most scalable architectural paradigms
- Part of cost-based query optimization (compile-time)
- Effective parallel query execution

Introduction and Motivation (cont.)

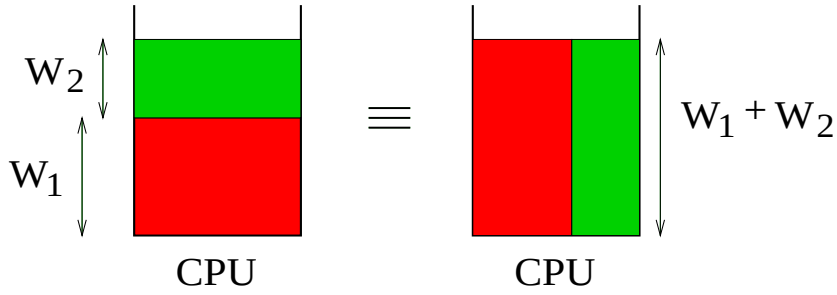
Sources of Complexity

- Various forms of intra/inter-operator parallelism
- Cost Model must account for communication overheads
- "Multi-dimensional" resource needs of DB operators (opportunities for resource sharing)
- Resources: *Time-Shared* (e.g, CPU(s)) or *Space-Shared* (e.g., memory)

- *NP-hard*, even for simple "one-dimensional" cases
- Our prior work [SIGMOD'96]: near-optimal solutions for TS resources *only*

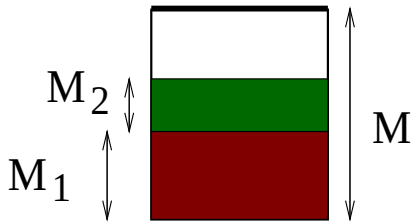
Goals Incorporate SS dimensions; Effective scheduling heuristics; Implications for Query Optimization

Time-Shared vs. Space-Shared



Time-Shared (TS) Resources

- Sliced between operators at negligible overhead
- Effective time can be stretched over execution interval



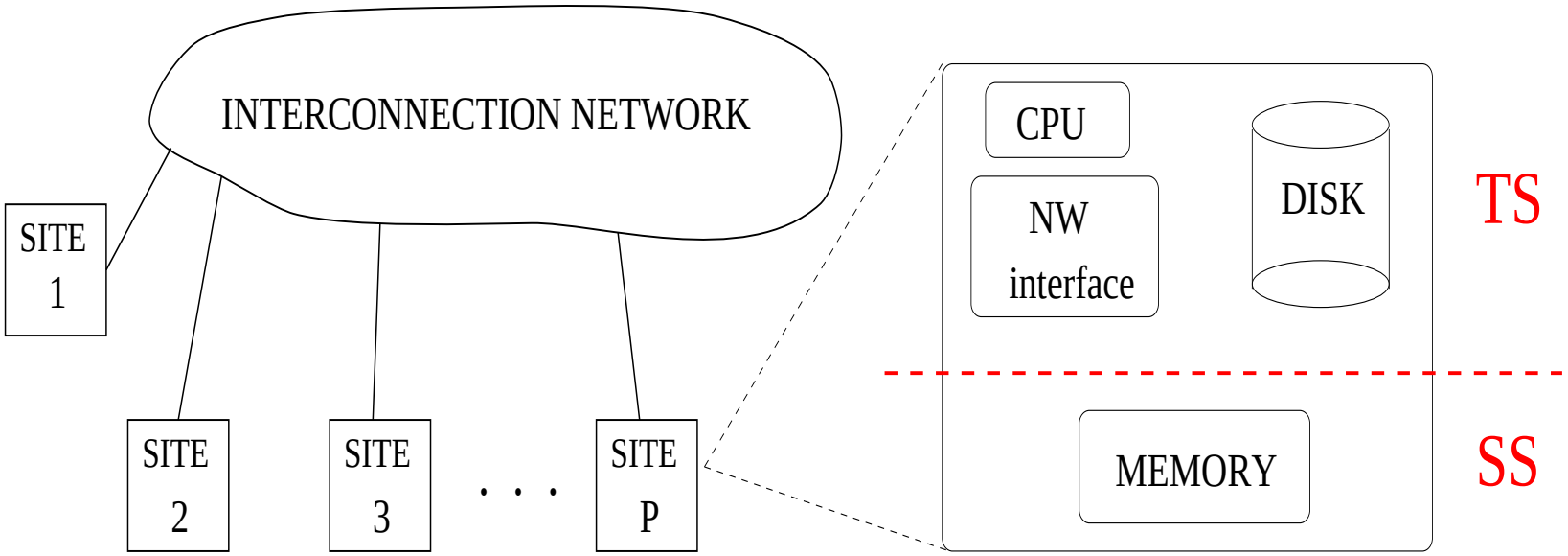
$M_1 + M_2 \leqslant M$

Space-Shared (SS)Resources

- Slicing implies prohibitive overhead
- Limited capacity that cannot be exceeded by executing operators

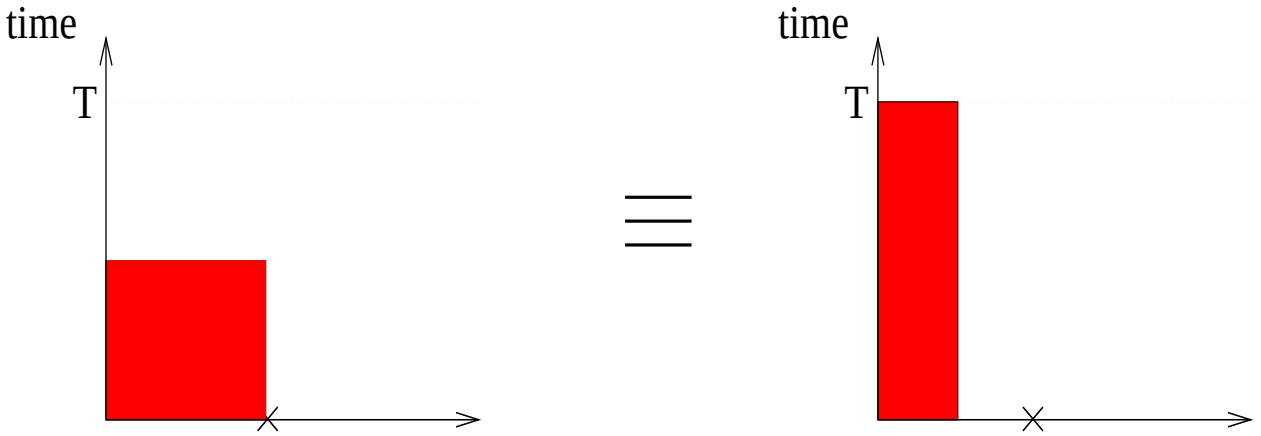
System Model

- Identical sites consisting of d TS and s SS resources



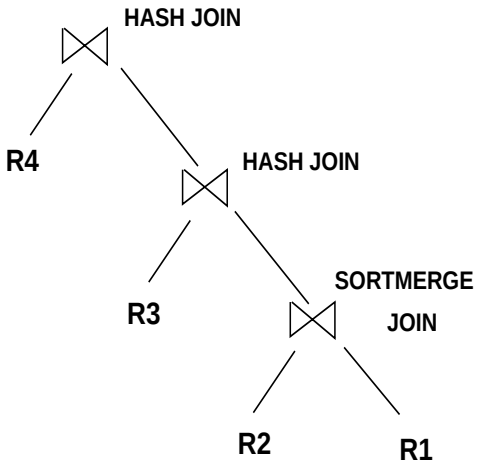
Main Assumptions

- No time-slicing overhead for TS resources
- Uniform TS resource usage

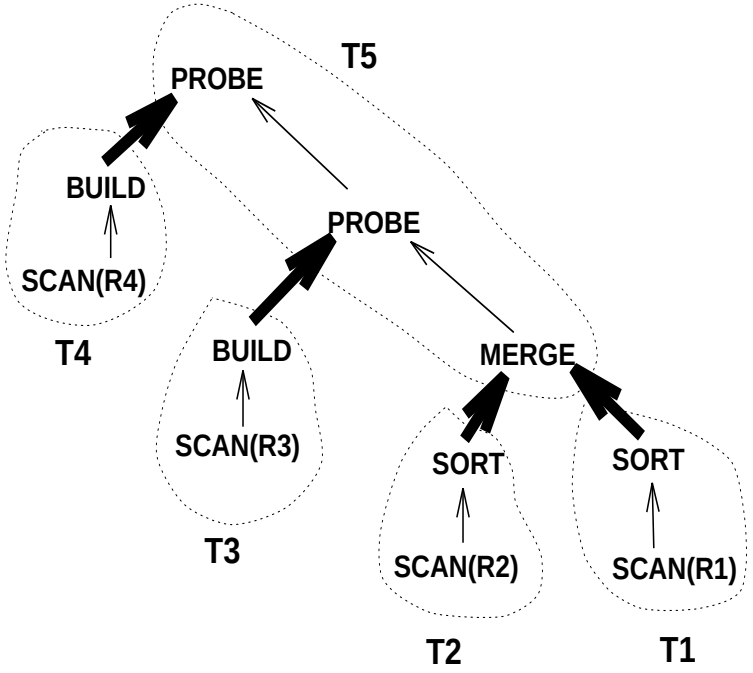


- Constant SS resource demands

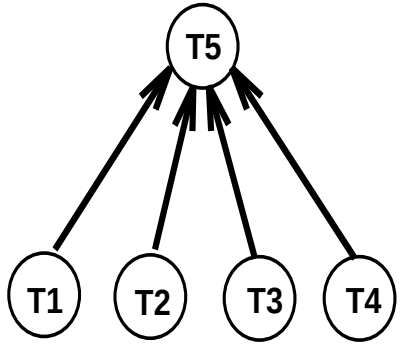
Query Plan Representation and Forms of Parallelism



JOIN TREE

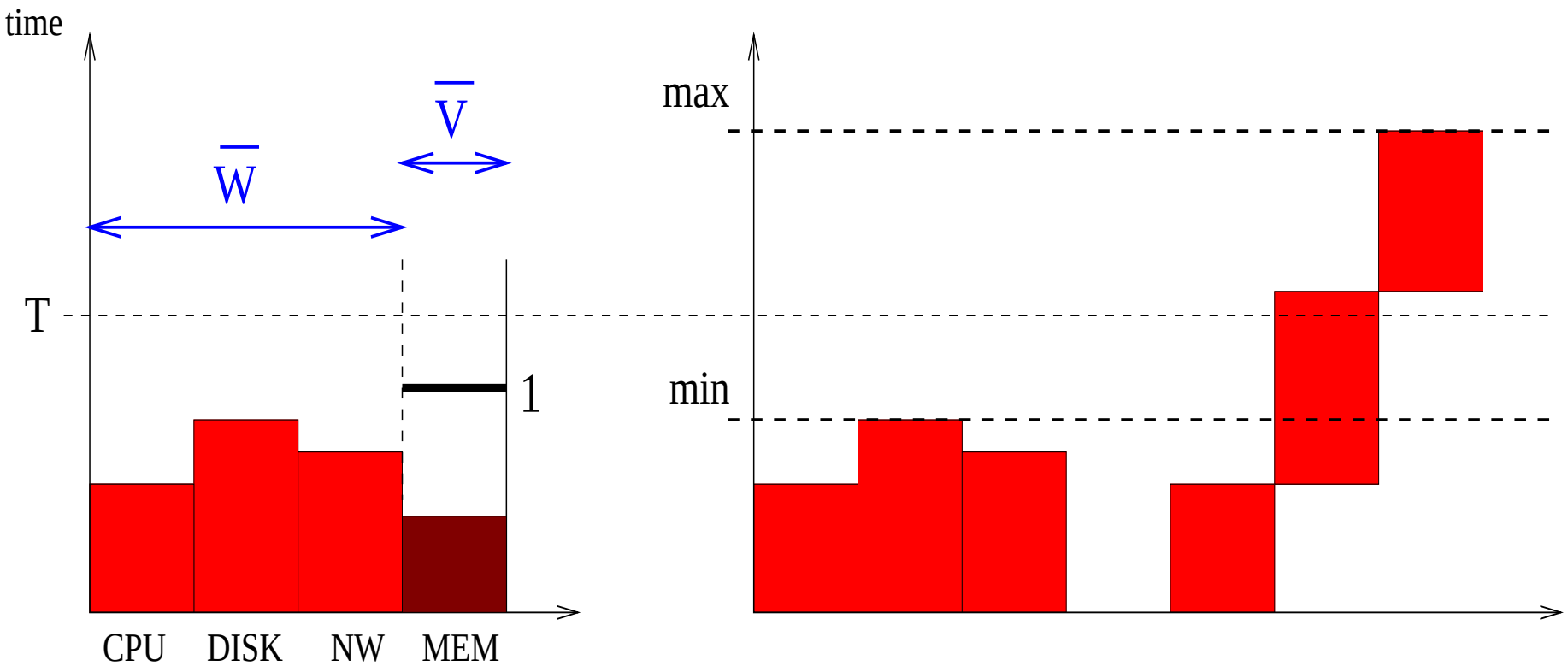


OPERATOR TREE



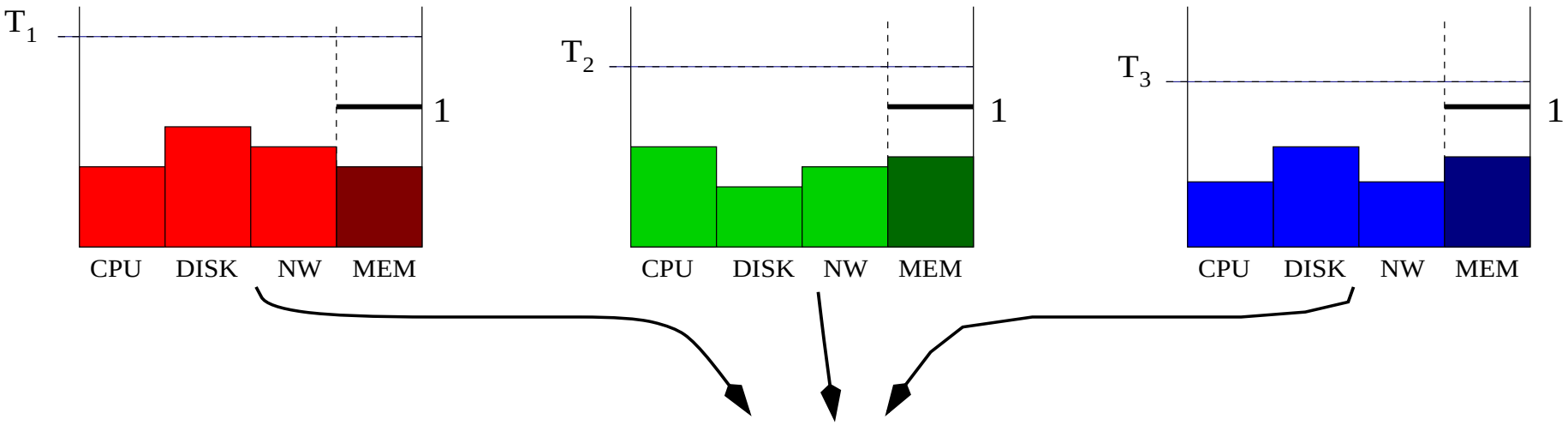
QUERY TASK TREE

Operator Cost Model

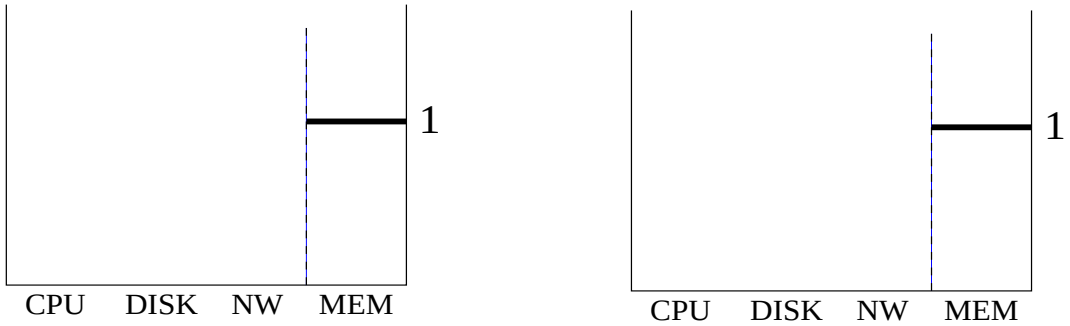


- Estimate *Work Vector* ($\overline{W}$) and *Demand Vector* ($\overline{V}$) components using conventional cost models and DB statistics
- Stand-alone execution time T = f($\overline{W}$, overlap), where $\overline{W}$ depends on $\overline{V}$

A Lower Bound on Response Time

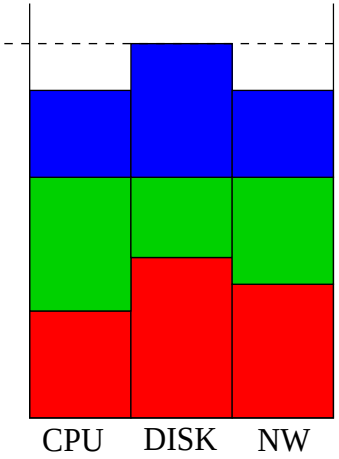


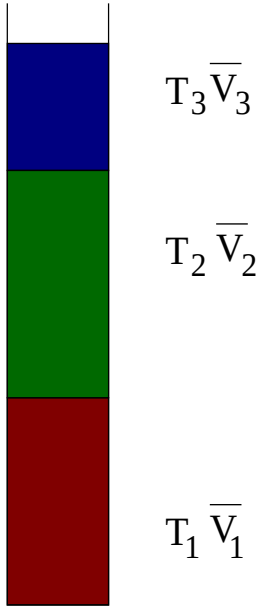
2 sites :



A Lower Bound on Response Time

- max { T_1 , T_2 , T_3 }: ‘‘critical path’’ of parallel execution





- $\dfrac{\text{max component } (\overline{W}_1 + \overline{W}_2 + \overline{W}_3)}{2}$

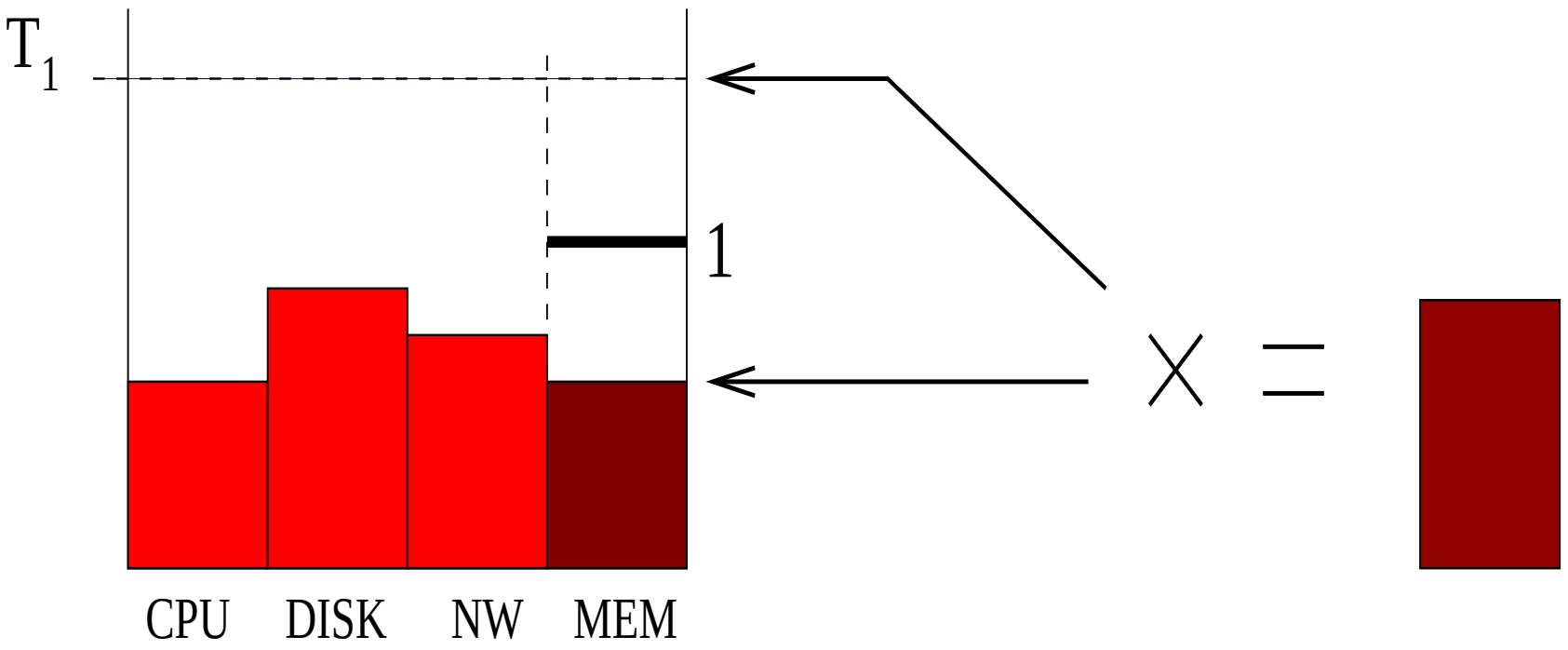
‘‘average work’’ per site (TS resources)

- $\dfrac{\text{max component } (T_1 \overline{V}_1 + T_2 \overline{V}_2 + T_3 \overline{V}_3)}{2}$

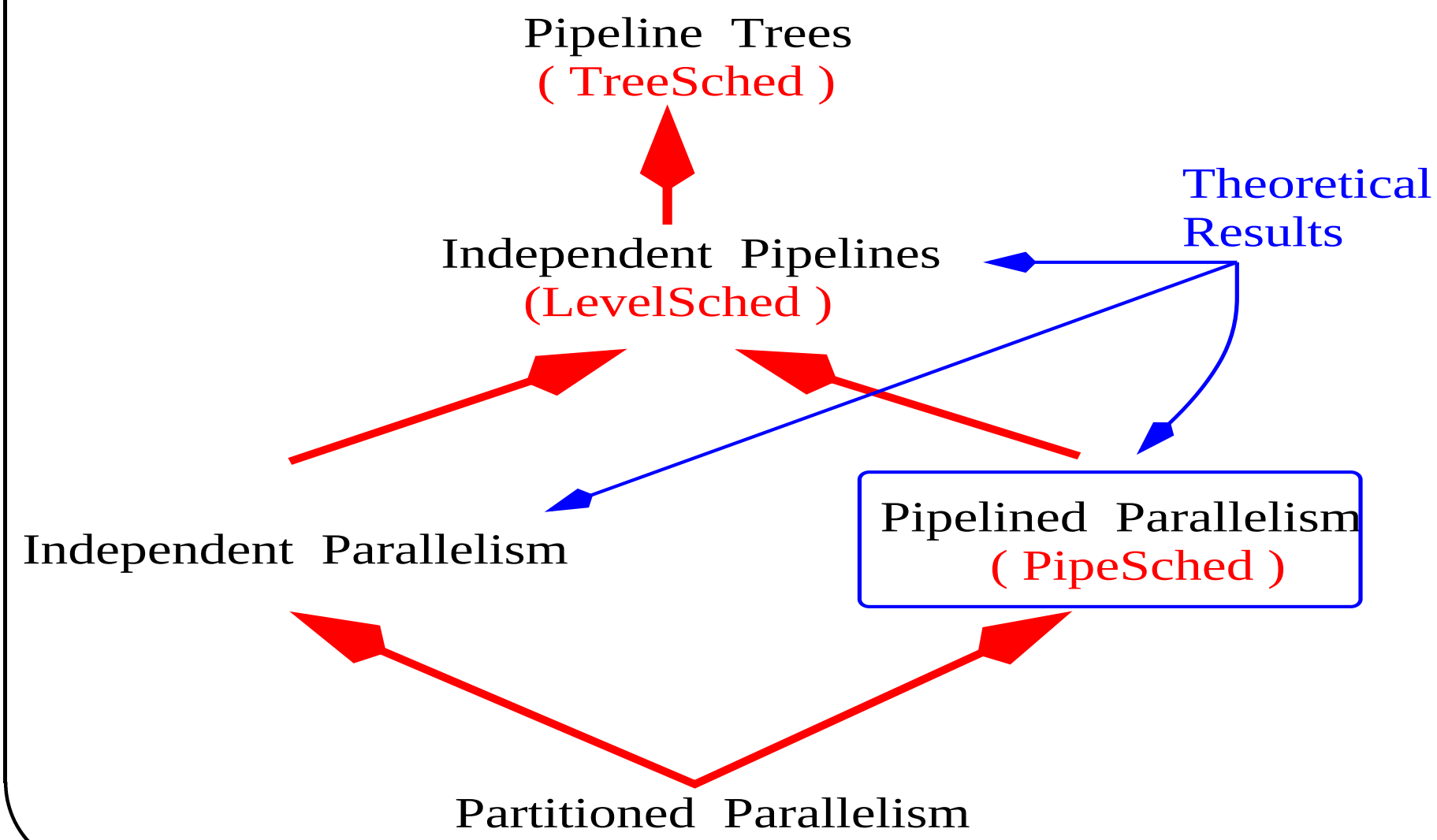
‘‘average volume’’ per site (SS resources)

$$LB = \max \{ \text{critical path} , \text{average work} , \text{average volume} \}$$

Operator Volume

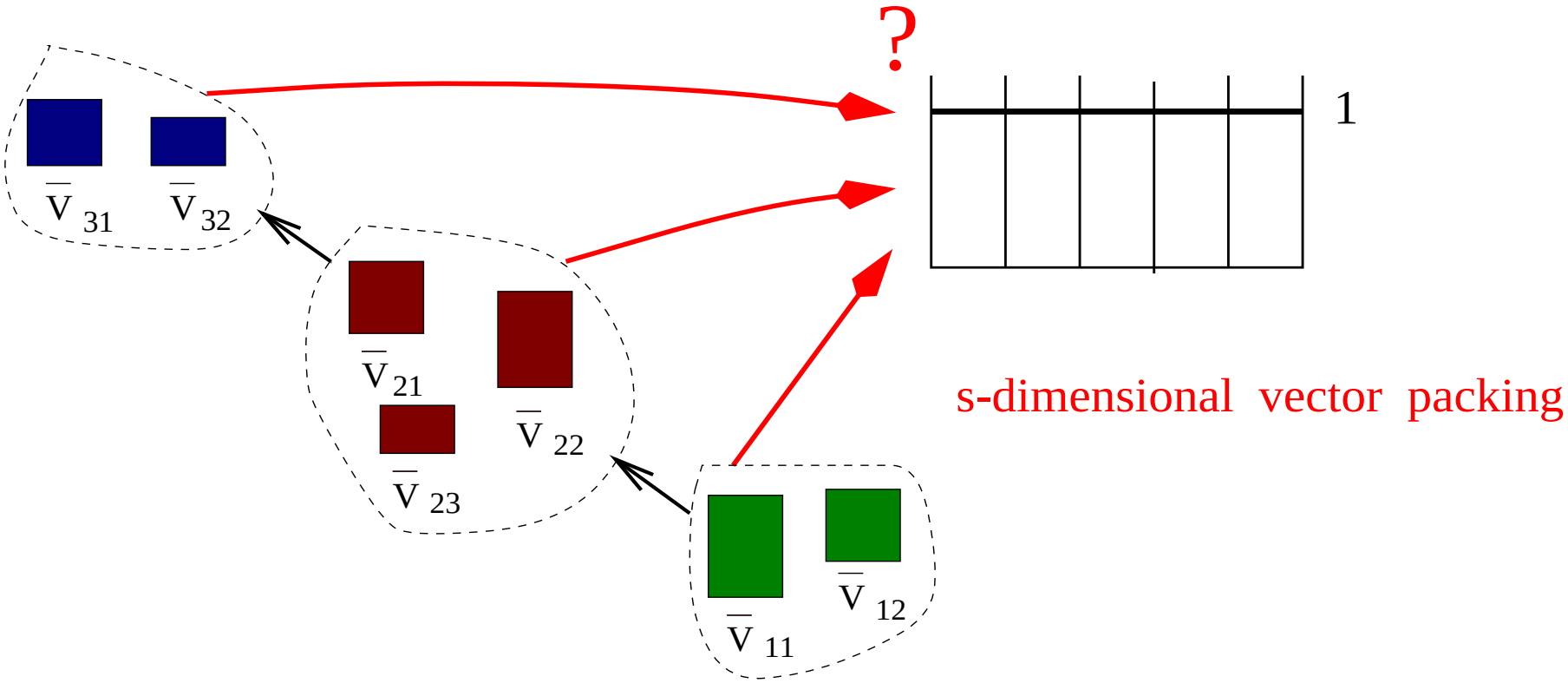


Scheduling Algorithms



Pipelined Parallelism

- Co-scheduling requirement: NP-hard decision problem!



- λ-*granular clone*: all components of $\overline{V}$ are $\leq \lambda$ (≤ 1)
- Simple sufficient condition for schedulability of pipelines of λ-granular clones

Pipelined Parallelism (cont.)

Algorithm PipeSched: fast List-Scheduling Heuristic (à la Graham) for a schedulable pipeline

1. Sort all clones in decreasing order of

$$work\ density\ =\ \max\{\overline{W}\}\ /\ \max\{\overline{V}\}$$

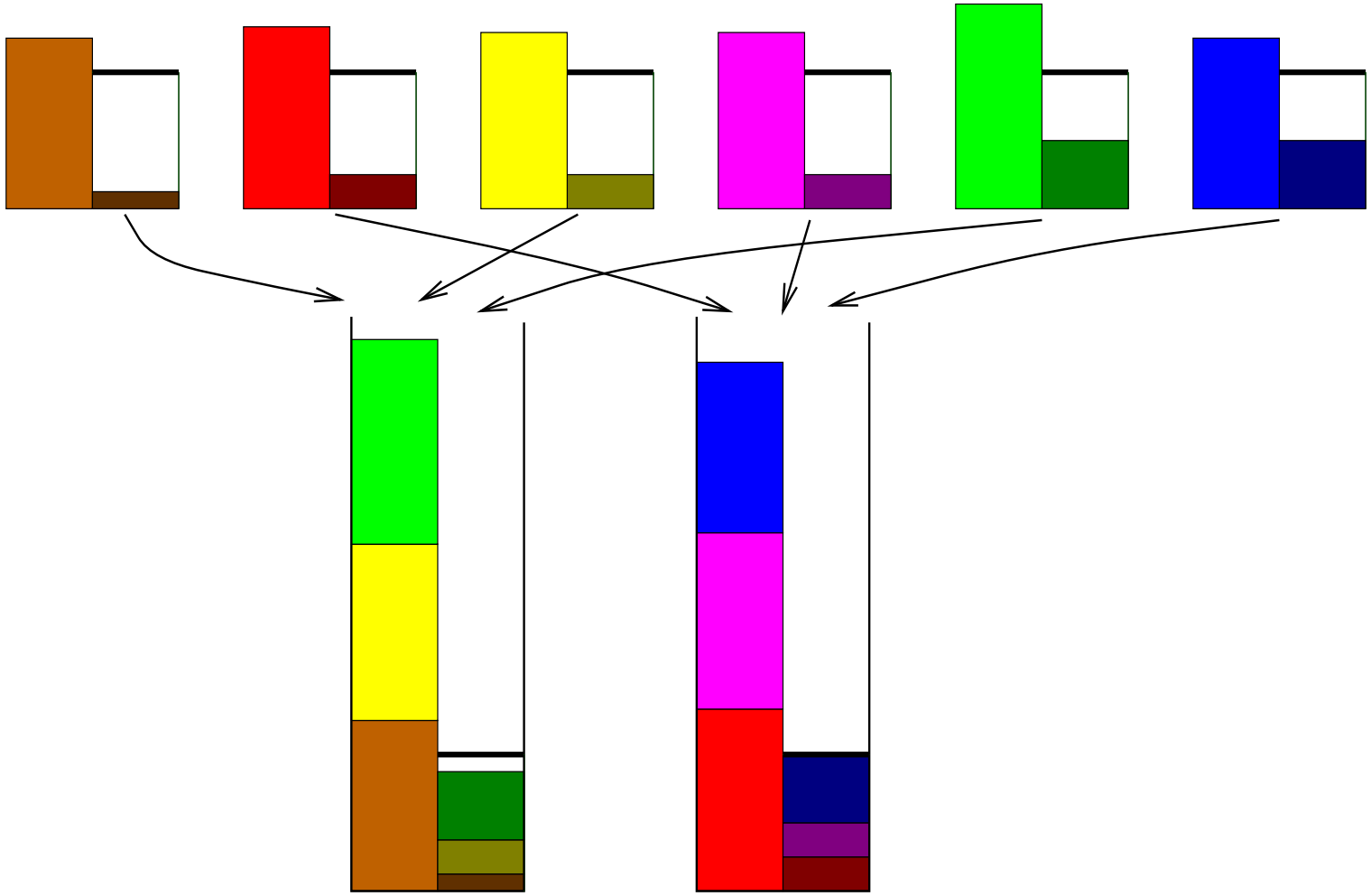
2. In that order, place each clone in the eligible site with the least amount of work

Theorem: Given a schedulable λ-granular pipeline, PipeSched produces a schedule that is within $d\left(1+\frac{s}{1-\lambda}\right)+1$ of optimal

A Simple Example ($d = 1,\ s = 1$)

1. Sort

2. Place



Independent Pipelines

Algorithm LevelSched**:**

1. Group the pipelines into maximal schedulable subsets
2. Schedule each subset in a separate level ("shelf") using PipeSched

Theorem: LevelSched is within $O\left(\frac{s(d^2+s)}{1-\lambda}\right)$ of optimal

Experimental Results

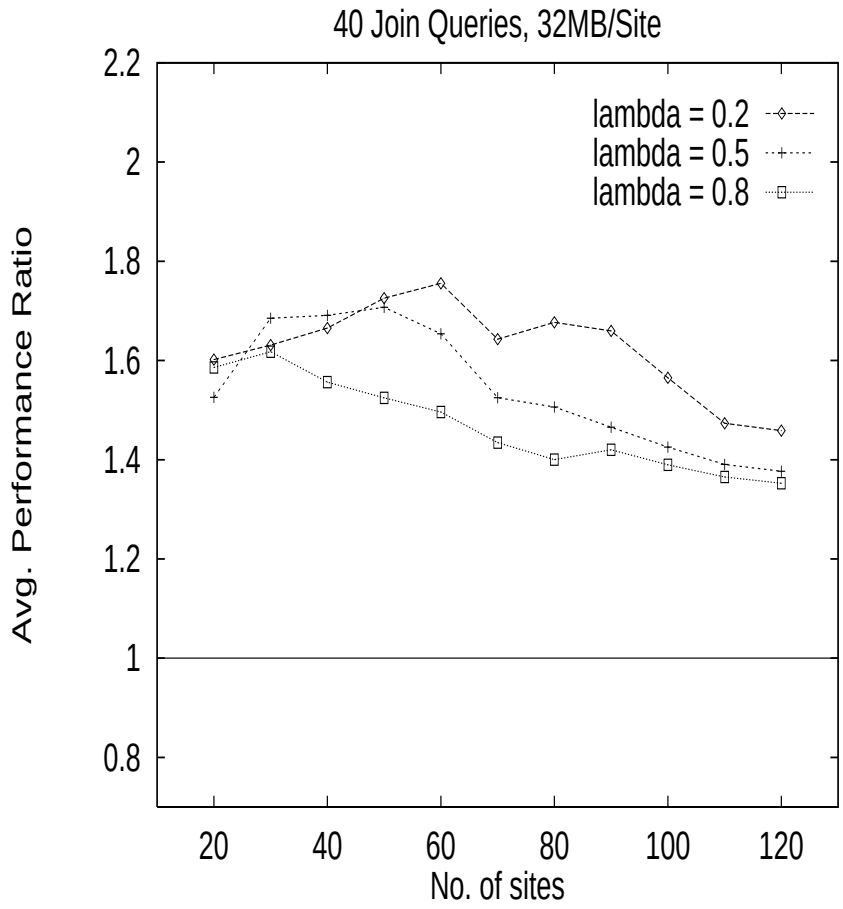
- TreeSched vs. lower bound on optimal response time (LB)
- *Average* effectiveness based on *cost model*

Experimental Assumptions

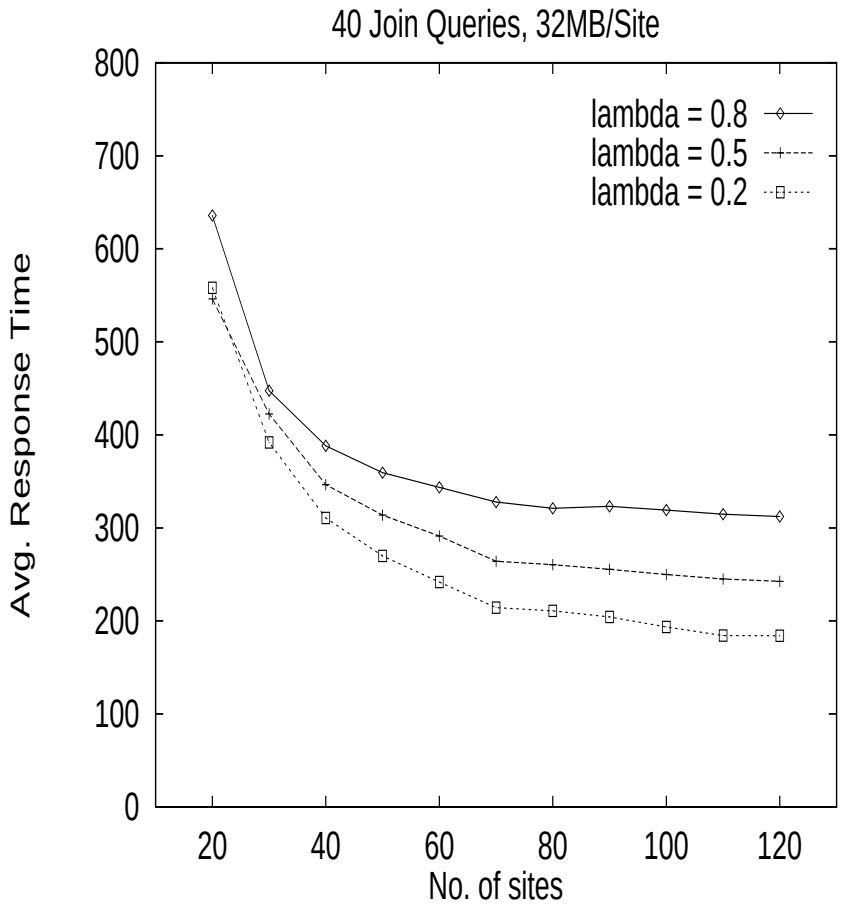
- Only simple hash-joins (one-phase pipelines)
- No execution skew (except for startup cost)
- Uniform resource overlap (single parameter)

Experimental Results (cont.)

- TreeSched / LB

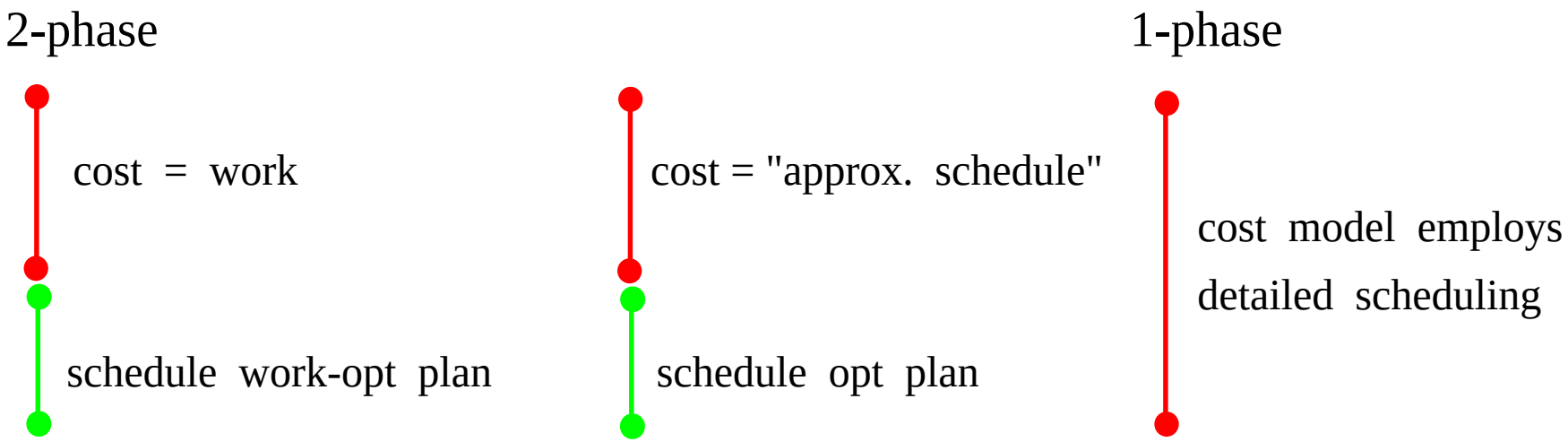


- TreeSched



Parallel Query Optimization

Cost Model: "plan quality" by abstracting execution environment



- [Ganguly et al., PODS'96]: max { critical path, average work }
- Our results: *average volume* captures the effects of SS resources
- $<$ *critical path, average work , average volume* $>$: crucial bulk parameters of a parallel plan for efficient optimization

Ongoing and Future Work

- *Malleable* memory demands
- Different degrees of "Time-Shareability" (e.g., disk vs. CPU)
- Evaluate effectiveness of scheduling algorithms (using ZetaSim)
- Validate proposed optimizer cost model

Partitioned Parallelism

Degree of Parallelism? Tradeoff...

- Finer clones: satisfy SS resource demands, better load balancing
- Coarser clones: reduce communication costs

Independent Parallelism

- *Compatible clones*: total SS resource demands ≤ 1
- Schedule: partition clones into compatible subsets; map onto sites
- Response time in a compatible partition is determined by TS components
- Algorithm OpSched: simple near-optimal heuristic

Query Task Trees: Handling Data Dependencies

Algorithm TreeSched:

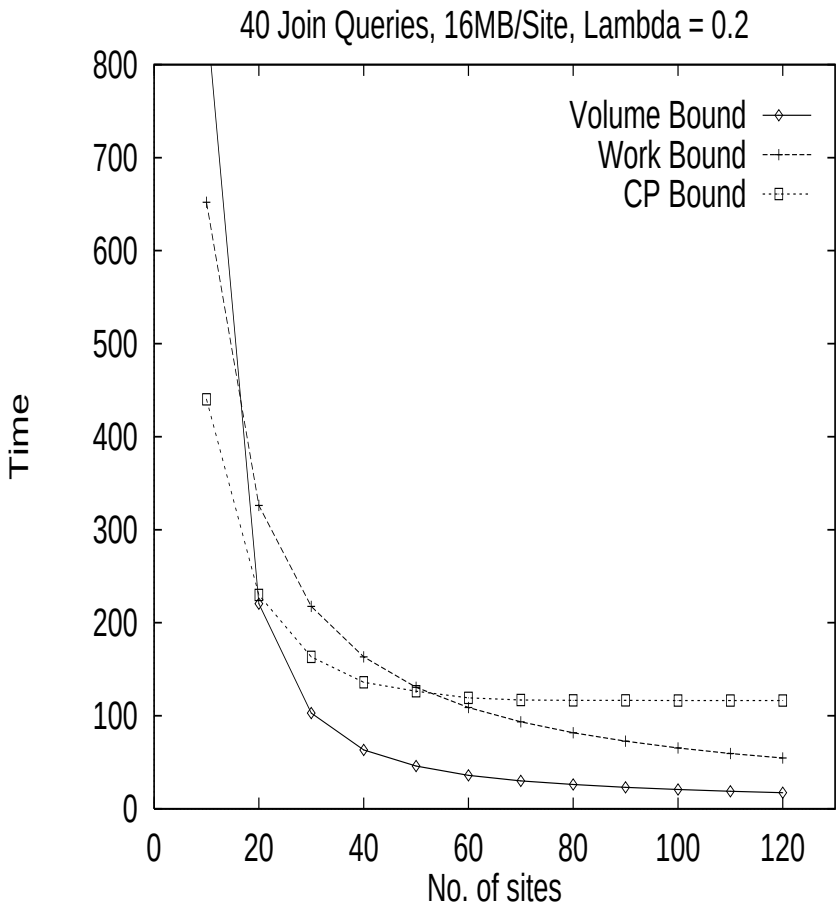
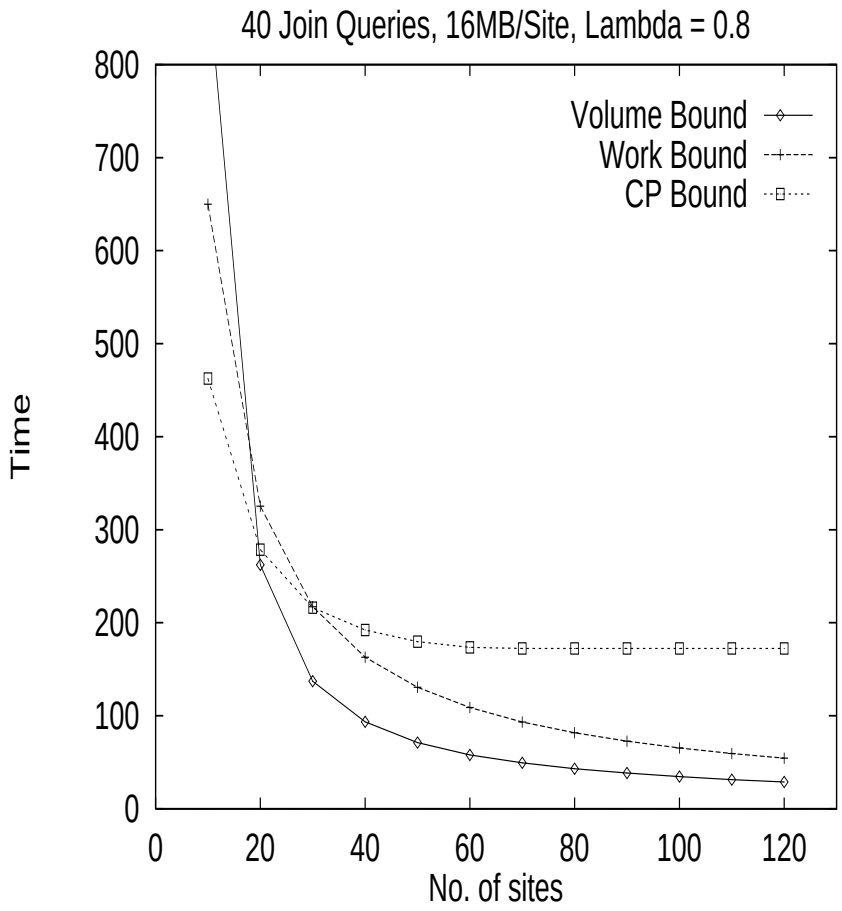
- Maintain dynamic list of enabled ("ready") pipelines starting from leaves
- Ensure resource dependencies are maintained (e.g., `build` $\rightarrow$ `probe`)
- Same ideas can be applied for scheduling tasks from multiple queries *on-line* (inter-query parallelism)

Experimental Testbed

- Sites with d=3 (CPU, Disk, NW interface) and s=1 (Memory)
- Tree queries 10,20,30,40,50 joins (20 random query graphs/size)
- Memory/site: 16MB - 64MB, λ = 0.1 - 0.9
  Overlap: 10% - 70%, Coarse Granularity: 0.3 - 0.9
- Performance metrics: Average performance ratio, Average schedule response time

Experiment Parameter Settings

Configuration/DB Catalog Params	**Value**
Number of Sites	10 – 120
CPU Speed	10 MIPS
Memory per Site	16 – 64 MB
Effective Disk Service Time per page	5 msec
Startup Cost per site (α)	15 msec
Network Transfer Cost per byte (β)	0.3 μsec
Fudge Factor (F)	1.4
Tuple Size	128 bytes
Page Size	32 tuples
Relation Size	10^4 – 10^6 tuples
CPU Cost Parameters	**No. Instr.**
Read Page from Disk	5000
Write Page to Disk	5000
Extract Tuple	300
Hash Tuple	100
Probe Hash Table	200

Experimental Results (more...)

- Lower Bound Components, $\lambda = 0.2$, 0.8