A Distributed System for Genetic Linkage Analysis

Mark Silberstein, Dan Geiger, Assaf Schuster

Distributed, High-Performance and Grid Computing in Computational Biology.

International Workshop, GCCB 2006, Eilat, Israel

Superlink-Online

- Distributed system on grid of computers.
  - for computing multipoint LOD scores of large pedigrees.
  - Based on serial algorithm Superlink.
  - High performance via efficient parallelization.
  - Available online:
    - Near-interactive response times for small problems while serving massively parallel ones at the same time.

Outline

- Scheduling algorithm
- Problem parallelization
- Current deployment in practice
- Performance evaluation

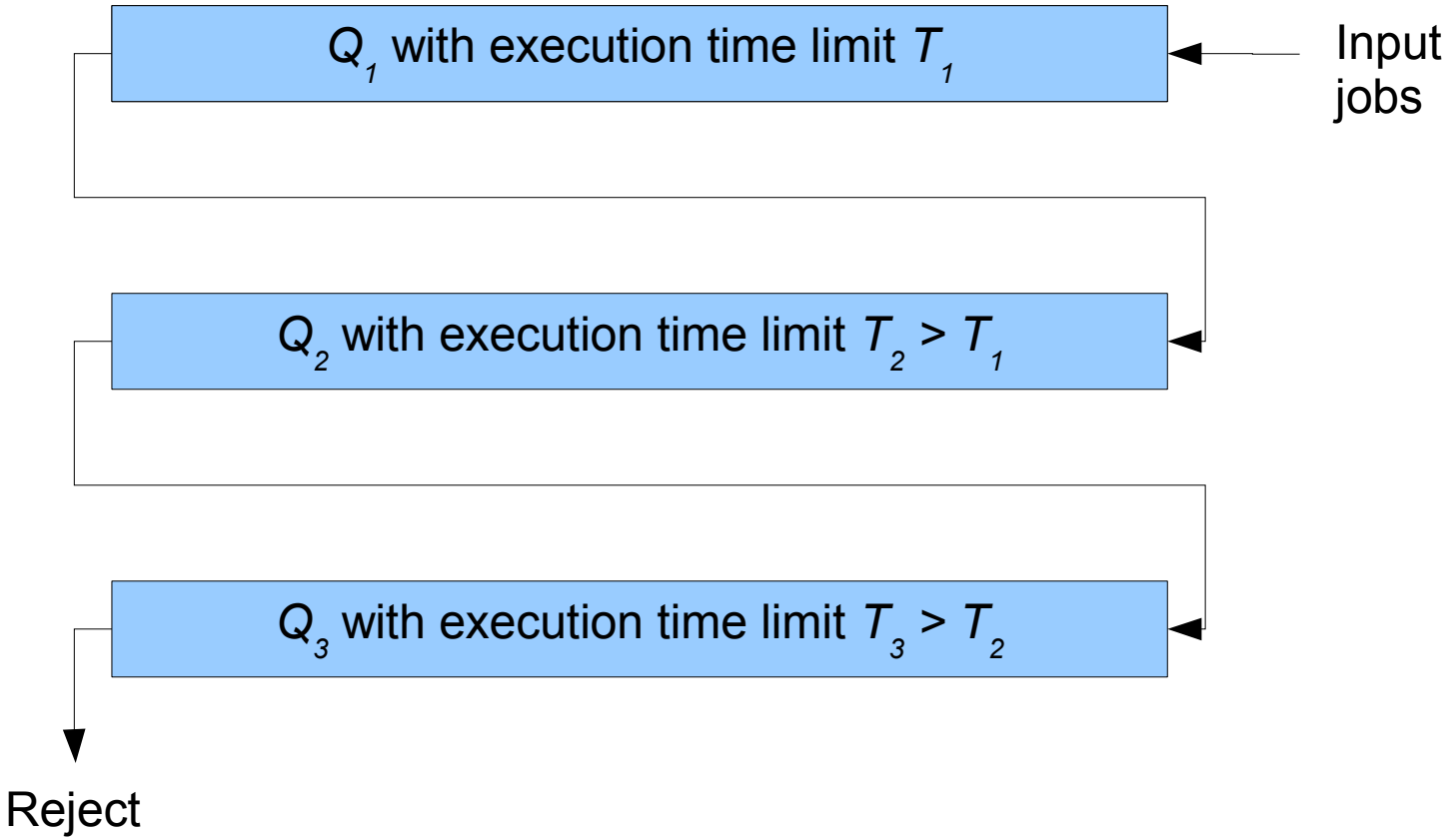
Grid Execution Hierarchy Scheduling

- Two complementary components:
  1) Impose hierarchy among several grids.
     - Based on grid performance characteristics.
  2) Schedule tasks on the hierarchy.
     - Problem of finding proper hierarchy level.

Grid Hierarchy

- Classify grids into levels.
  - According to performance:
    - Execution overhead, amount of resources, etc.
- Flexible number of levels.
  - Based on expected distribution of task complexities.
- At each hierarchy level:
  - link to set of one or more job queues, connected to corresponding grids.

Multilevel Feedback Queue (MQ)



- Fails to provide fast response time to short tasks if long task is present, since queues are FCFS.

Avoiding Hierarchy Level Mismatch

- Impose task complexity limit C_i for queue Q_i :
  - Optimistic assumption $C_i = T_i * (N * P * \beta)$
  - For each job *j* arriving at queue Q_i :
    - Reserve time $\alpha * T_i$ for computing complexity estimate B_j .
    - If $B_j > C_i$, migrate job *j* to queue Q_{i+1}, where complexity is reestimated (with more resources).

Multiple Grids at the Same Level

- A hierarchy level can have more than one grid attached to it:
  - Each grid has one associated queue.
  - Queues periodically sample status of all other queues at this level.
  - Apply heuristics to migrate tasks and balance workload.

Outline

- Scheduling algorithm
- Problem parallelization
- Current deployment in practice
- Performance evaluation

LOD Score Computation

- In general, expression of the following form:

$$\sum_{x_1}\sum_{x_2}\cdots\sum_{x_n}\prod_{i=1}^{m}\Phi_i(X_i)$$

- Complexity estimation:
  - Stochastic greedy anytime algorithm.
  - Yields elimination ordering and upper bound on complexity.
  - Improves over time.

Parallelization

- Two requirements:
  - Subtasks cannot communicate or synchronize.
  - Must tolerate frequent failures of subtasks in grid.
- Master-worker paradigm:
  1) Parallelize finding elimination ordering.
  2) Parallelize LOD score computation by recursively conditioning on summation variables x_1, x_2, ...
     - Until desired granularity is reached.

Choice of Granularity

- Trade-off in making subtasks smaller:
  - Improves load balancing and fault tolerance.
  - Increasing overhead inhibits performance.
- Specify max. allowable complexity threshold *C* such that:
  - Subproblems can run without interruption.
  - Available number of computers will be used.
  - Overhead will be less than 1% of running time.

Outline

- Scheduling algorithm
- Problem parallelization
- Current deployment in practice
- Performance evaluation

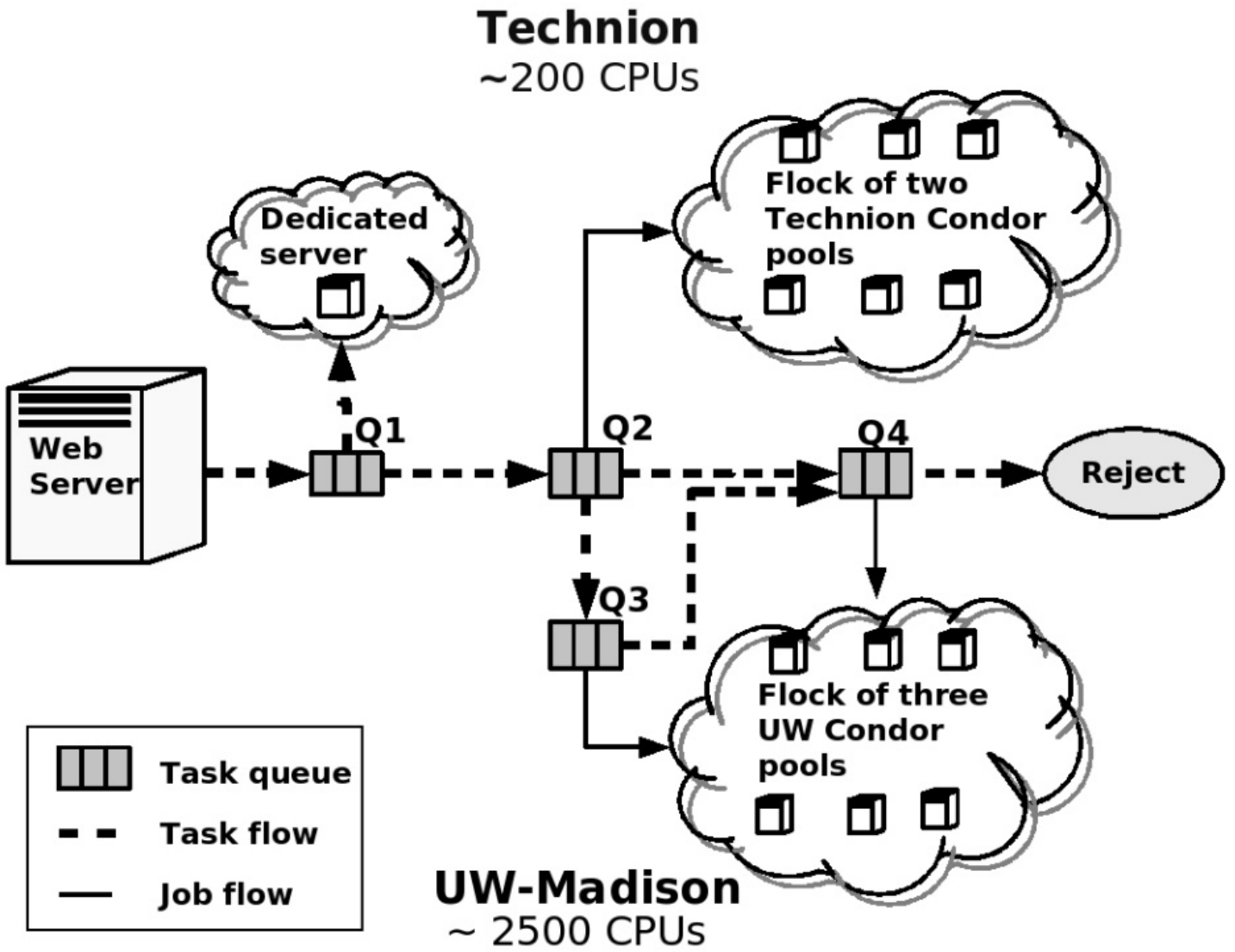
Superlink-Online Implementation

- Uses the Condor distributed batch system.
  - Opportunistic, handles job failures transparently.
  - Three stages of general master-worker application:
    - Parallelization of a task into independent jobs.
    - Parallel execution of these via Condor.
    - Generation of final results upon completion.
  - In this case, two master-worker applications:

    1) Parallel ordering estimation.

    2) Parallel variable elimination.

Superlink-Online Deployment as of 2006



Outline

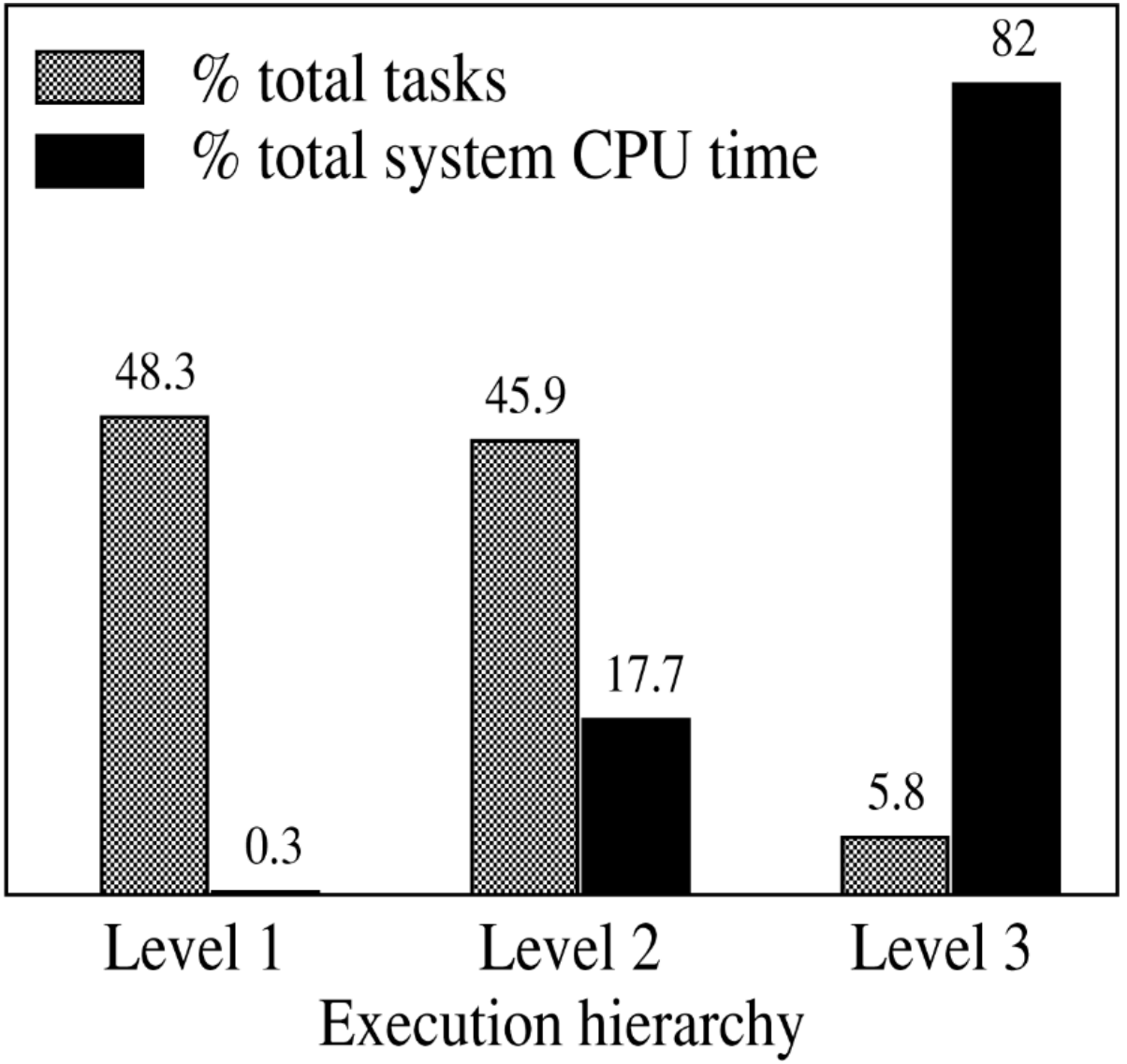
- Scheduling algorithm
- Problem parallelization
- Current deployment in practice
- Performance evaluation

Superlink-Online vs. Superlink 1.5

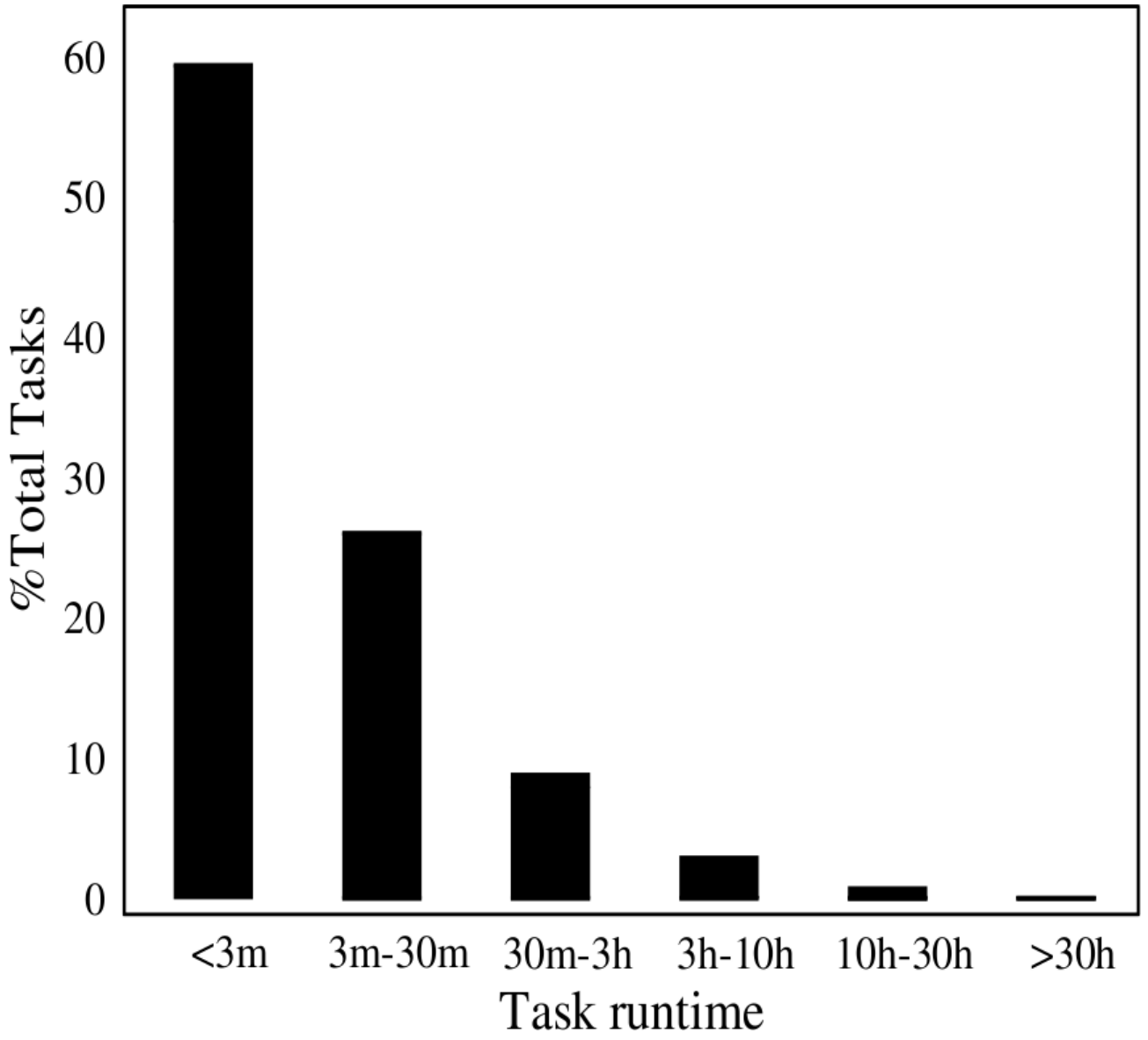
	Running time		#CPU used	
Input	SUPERLINK V1.5	SUPERLINK-ONLINE	Average	Maximum
1	5000sec	1050sec	10	10
2	5600sec	520sec	11	11
3	20hours	2hours	23	30
4	450min	47min	82	83
5	~300hours	7.1hours	38	91
6	297min	27min	82	100
7	~138days	6.2hours	349	450
8	~2092sec	1100sec	7	8
9	~231hours	3hours	139	500
10	~160days	8hours	310	360

Distribution of Task Complexity

- Statistics collected over 2300 tasks in 2nd half of 2005.
- 460,000 CPU hours (= 52.5 years).
  - 70% for 1971 successful tasks.
  - 3% wasted (failures and cancellations).
  - 27% timed out (with partial results).



Distribution of Real Task Runtime



Average Accumulated Time

- Time from submission to termination.
  - 70 % of overhead in Q_4 is delay due to waiting for other, longer tasks.