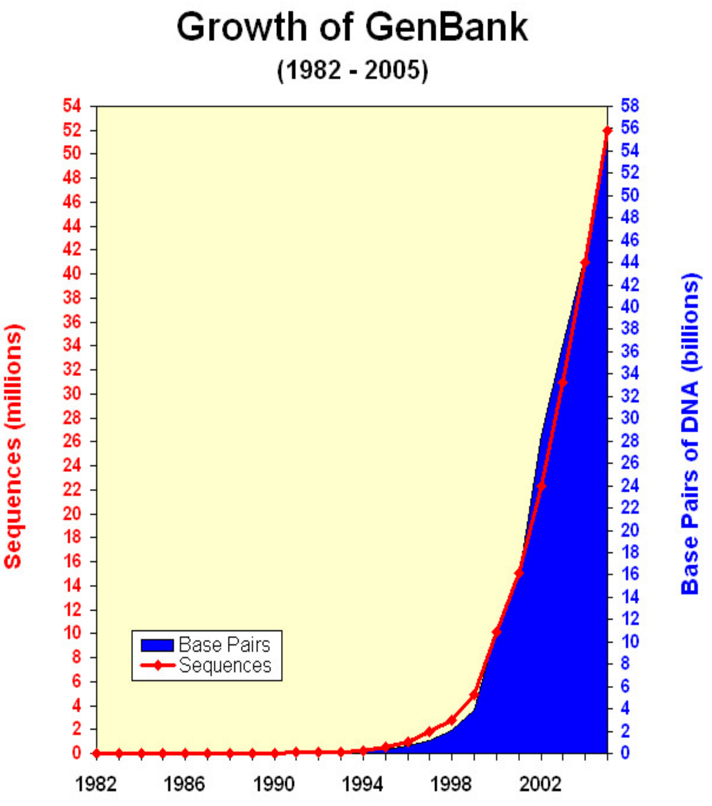
Adaptive Request Scheduling for Parallel Scientific Web Services

Heshan Lin
Xiaosong Ma
Jiangtian Li
Ting Yu
Nagiza Samatova

North Carolina State Universty
Oak Ridge National Laboratory

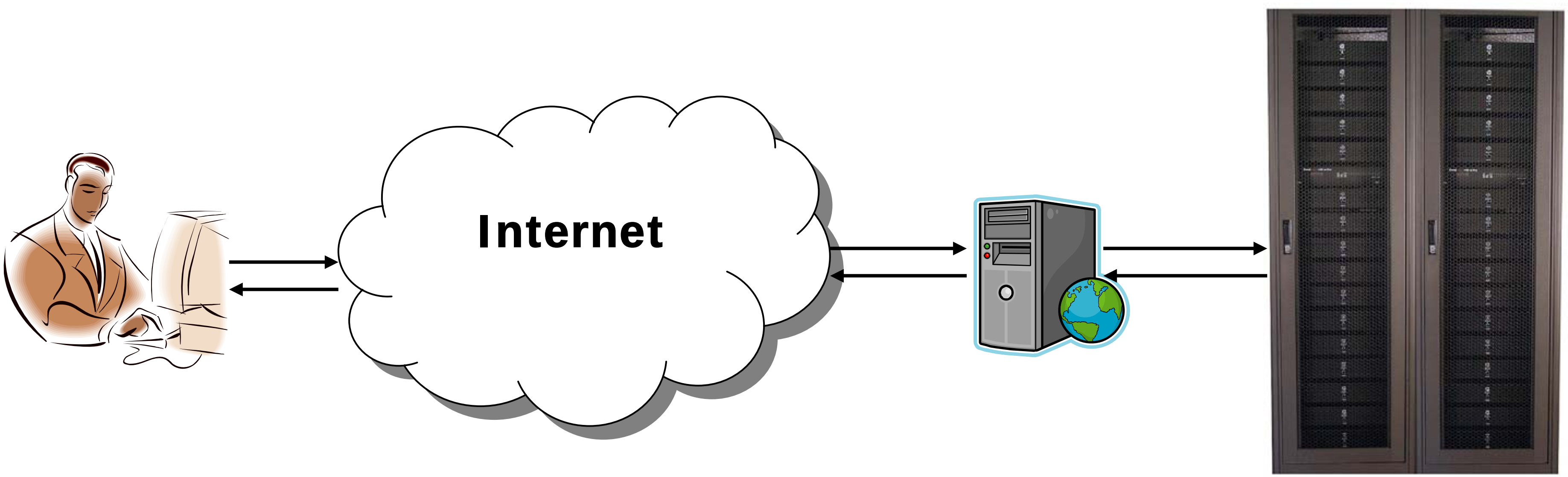
"Data Rich" Scientific Applications

- Scientists need to routinely process hundreds of GBs or TBs of data
  - Biology, cosmology, climate
- Public science data grow rapidly
  - E.g., GenBank size grows > 5 orders of magnitude in last 2 decades
- Storing, analyzing such data beyond capacity of personal computers



Scientific Web Services

- Increasingly popular to address data growth
  - Efficient sharing of
    - public data repository
    - high-end computing resources
  - Hiding parallel job management overhead



New Scheduling Context

- Characteristics of scientific requests
  - Compute-intensive: require processing on multiple processors
  - Data-intensive: accessing GBs to TBs of data
- Related scheduling studies
  - Content serving cluster web server: focusing on data-locality
  - Space sharing parallel job scheduling: focusing on parallel efficiency
- Needs computation- and data-aware scheduling algorithms

Our Contributions

- Two-level adaptive scheduling framework for scientific web services
  - Goal: to improve average request response time
  - Takes into account both data-locality and parallel efficiency
  - Automatically adapts to system loads and request patterns

- Case study: genomic sequence similarity search (BLAST) web server
  - Performance evaluation on real cluster

Road Map

- Introduction
- Background
- Scheduling design
- Experiment results
- Conclusions

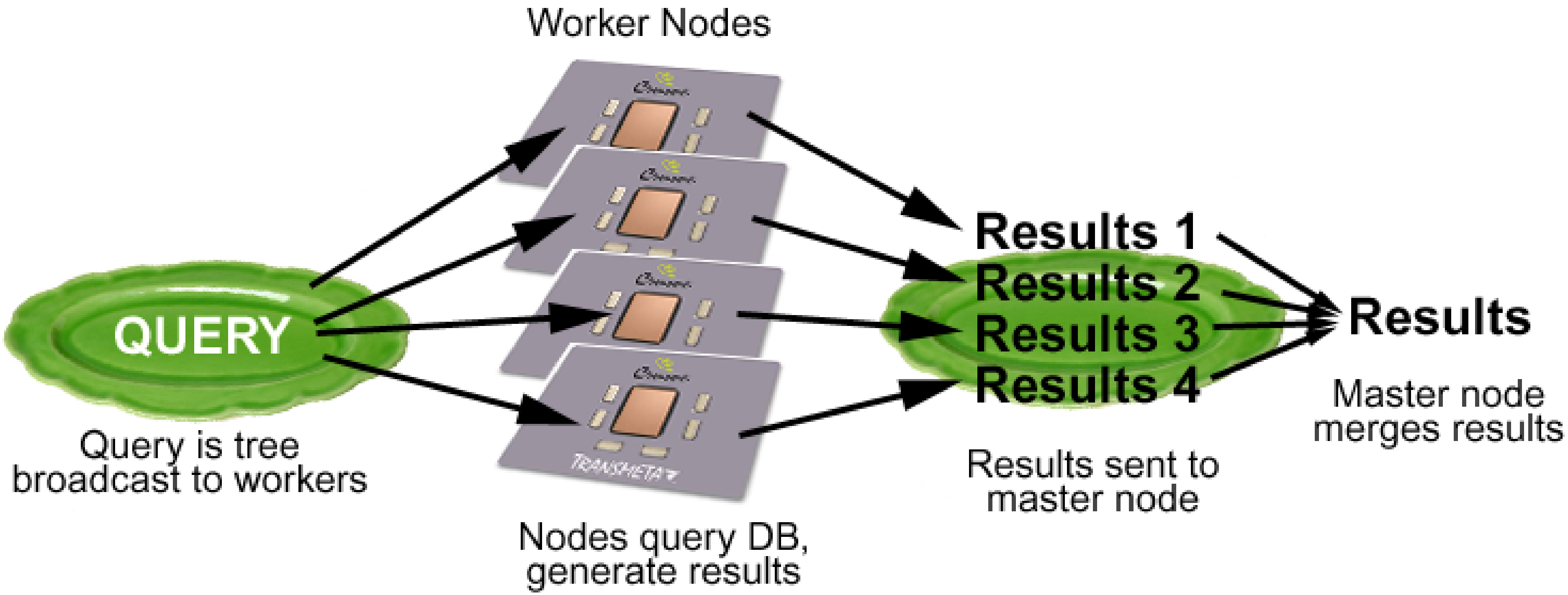
BLAST

- Routinely used in many biomedical researches
  - Search similarities between query sequences and those in sequence database
    - Predict structures and functions of new sequences
    - Verify experiment and computation results
- Analogous to web search engines (e.g. Google)

	Web Search Engine	BLAST
Input	Key word(s)	Query sequence(s)
Search space	Internet	Known sequence database
Output	Related web pages	DB sequences similar to the query
Sorted by	Closeness & rank	Score (Similarity)

Parallel BLAST

- Partition large DBs across multiple processors
  - mpiBLAST [Darling03, Lin05, Gardner06, Lin08]

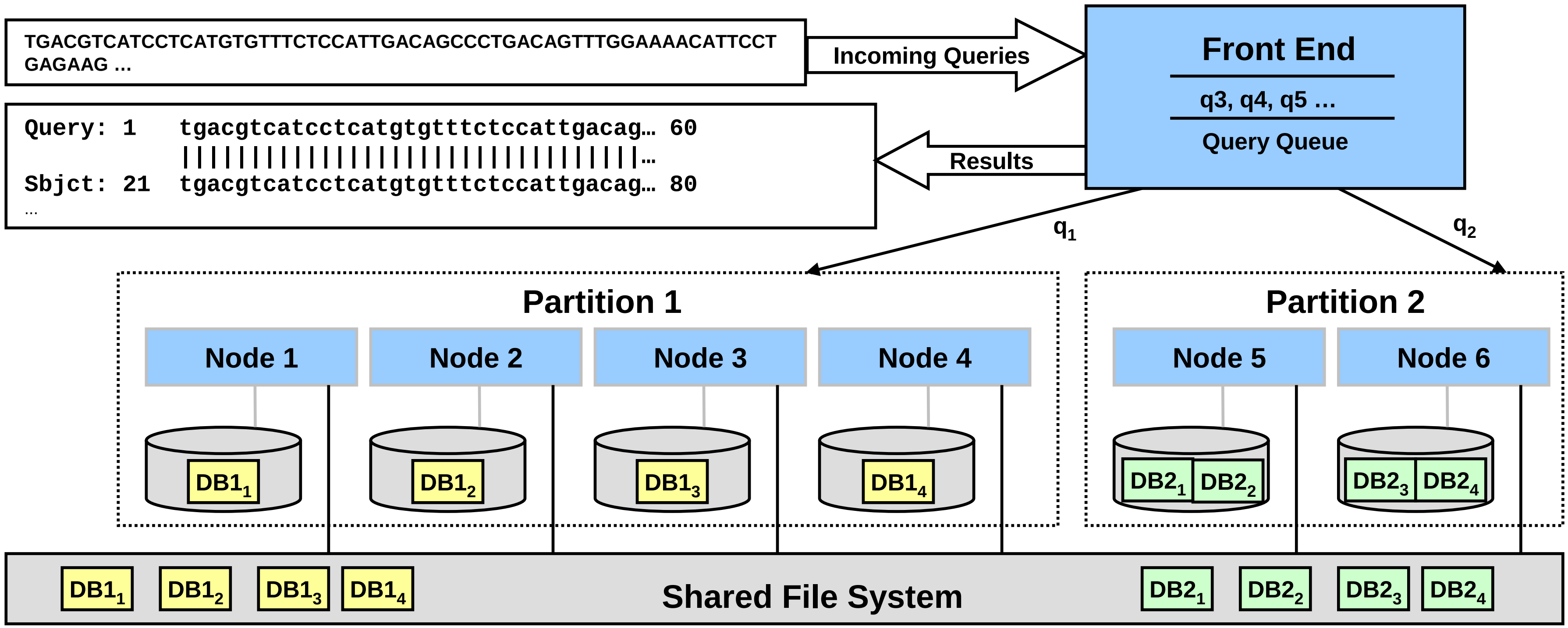


Road Map

- Introduction
- Background
- Scheduling algorithm design
- Experiment results
- Conclusions

System Architecture

- Front end node
  - Receives request and making scheduling decision
- Backend nodes
  - Perform parallel BLAST jobs



Overview

- Scheduling problem: find partition of cluster to service request
  - **How many processors to allocate?**
  - **And on which processors?**
  - Which database fragment(s) to search on each processor?

- Scheduling techniques
  - Efficiency-oriented scheduling
  - Data-oriented scheduling
  - **Challenge:** to automatically adapt to system loads and query patterns

Efficiency-Oriented Scheduling

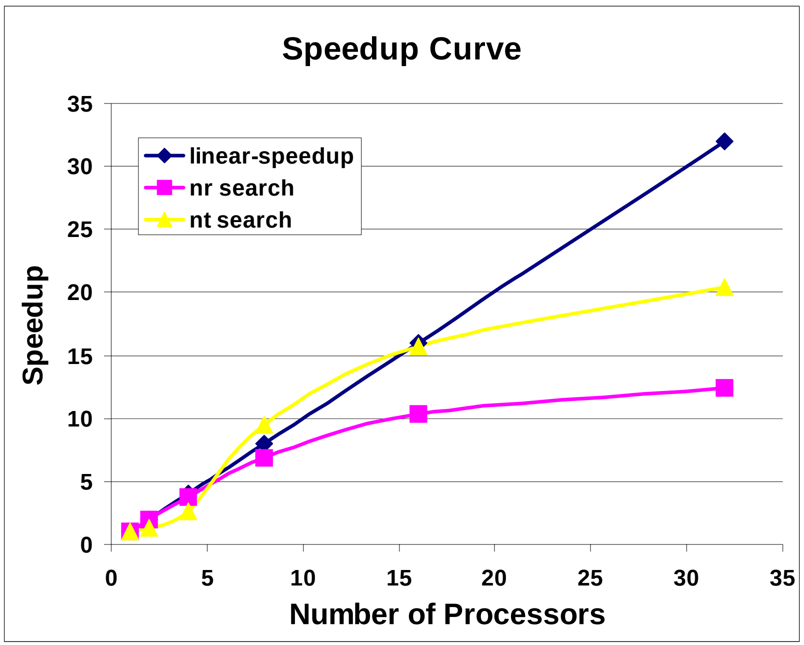
- Response time = wait time + service time
- Intuition
  - Partition size grows => speedup increases, efficiency decreases
  - When load light, use large partition size -> reduce service time
  - When load heavy, use small partition size -> reduce wait time
- MAP [Dandamudi99]
  - Compute partition size
  - S: number of jobs being serviced
  - f: adjustable parameter (0 <= f <= 1)

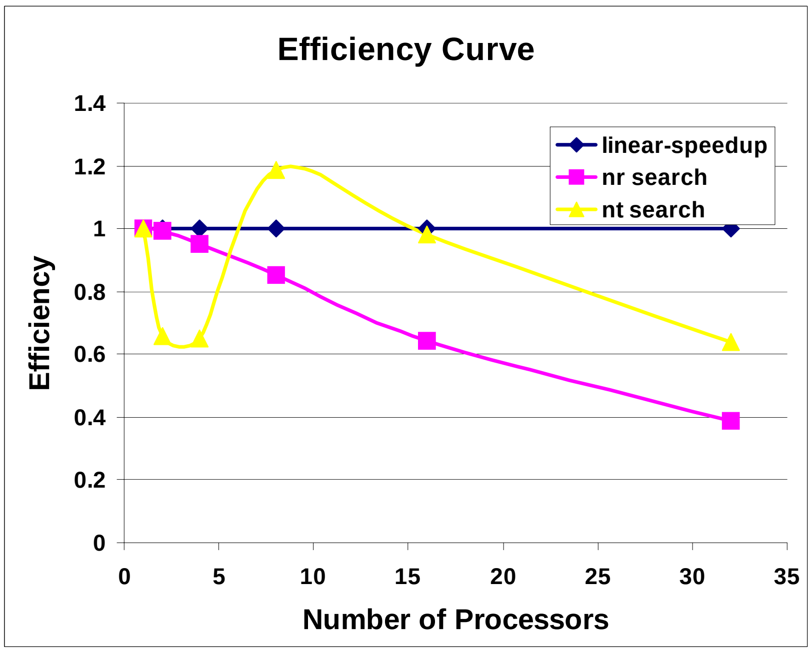
Speedup/#procs

$$partition\_size = Max(1, ceil(\frac{total\_processors}{queue\_length + 1 + f * S}))$$

Our Solution: RMAP

- Define a range of partition sizes [P_{min}, P_{max}] for each DB
  - P_{min}: smallest # procs whose aggregate memory can hold the database
  - P_{max}: saturation point of speedup curve





Data-Oriented Scheduling

- Given partition size ***p***, which processors should search next query?
- Naïve approach
  - FA (First Available): similar to batch job scheduling
    - Orders processors by rank, pick first ***p*** idle processors
    - Does not consider data locality
- LARD algorithm for cluster web servers [Pai98]
  - Intuition: assigns object request to processor that recently serviced it
  - Considers both data locality and load balance

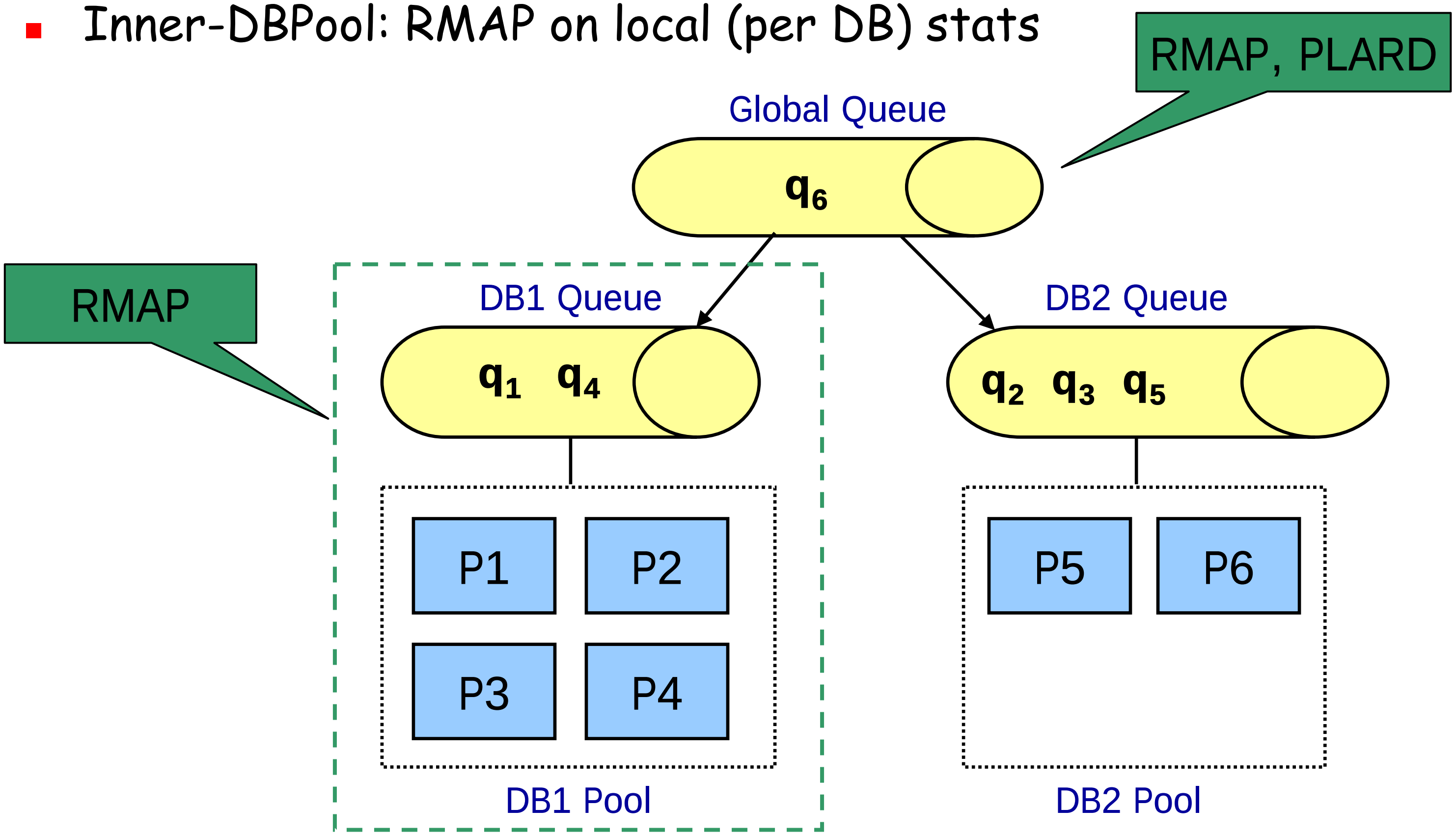
Data-Oriented Scheduling (cont.)

- What's new here?
  - Servicing a query requires co-scheduling of multiple nodes
  - A processor can only serve one query at a time

- ***Our solution: PLARD***
  - Multiple queues and processor pools
    - Per-database basis
  - Query assignment and load balancing among processor pools
    - Assign and migrate processors in groups

PLARD + RMAP

- Two-level scheduling decisions
  - Inter-DBPool: dynamically adjusting DB pool sizes guided by RMAP on global statss
  - Inner-DBPool: RMAP on local (per DB) stats



Road Map

- Introduction
- Background
- Scheduling design
- Experiment results
- Conclusions

Experiment Setup

- Input data
  - 5 NCBI sequence databases
  - Synthesized query trace, Poisson arrivals
    - 1000 randomly sampled query sequences (proportional to DB size)
- Backend cluster
  - 32 Xeon procs, Linux OS, Gigabit Ethernet

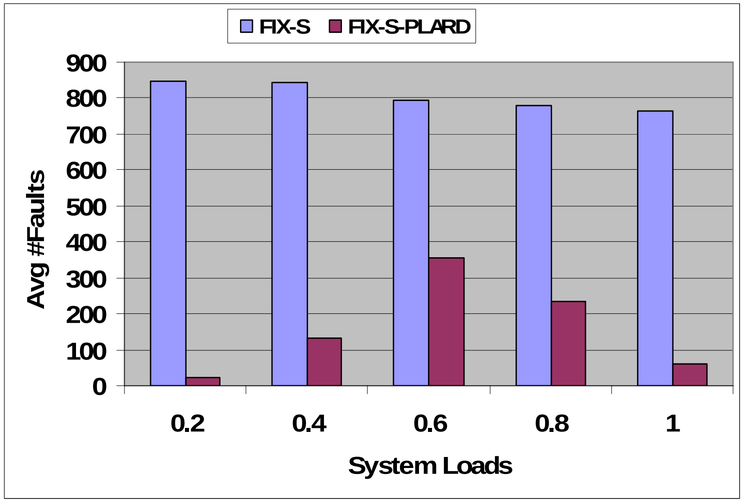
DB Name	Type	Raw Size	Formatted Size	P_{min}	P_{max}
env_nr	P	1.7GB	2.5GB	2	32
nr	P	2.6GB	3.0GB	4	32
est_mouse	N	2.8GB	2.0GB	2	16
nt	N	21GB	6.5GB	8	32
gss	N	16GB	9.1GB	8	32

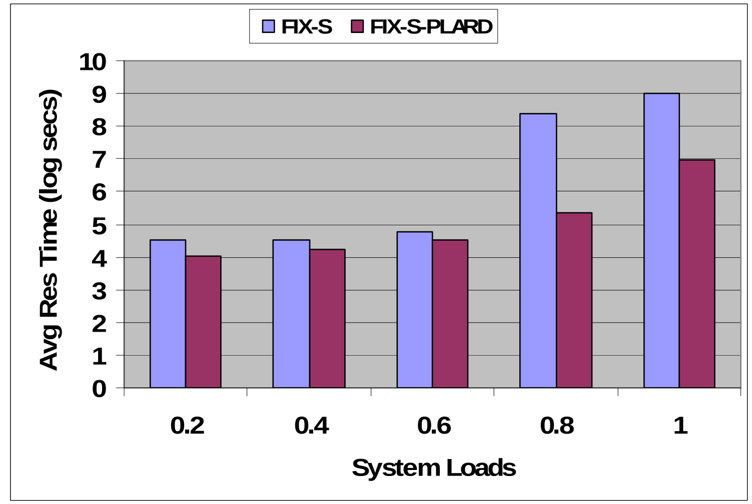
PLARD Impacts

Avg #Page Faults

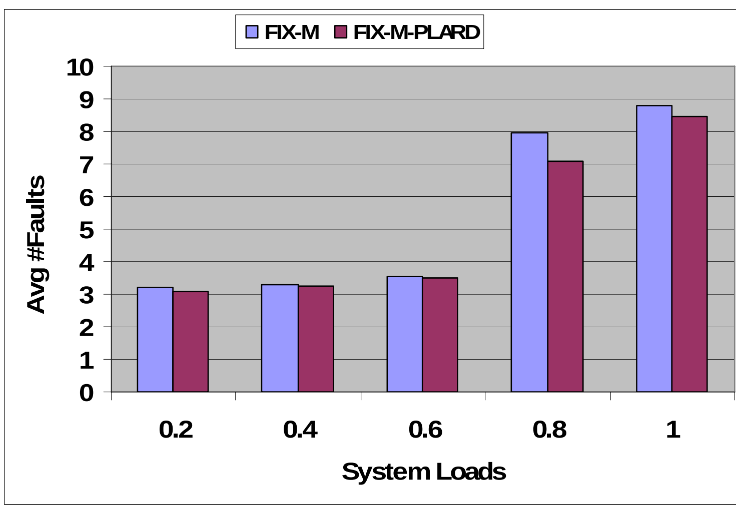
Avg Response Time (Log)

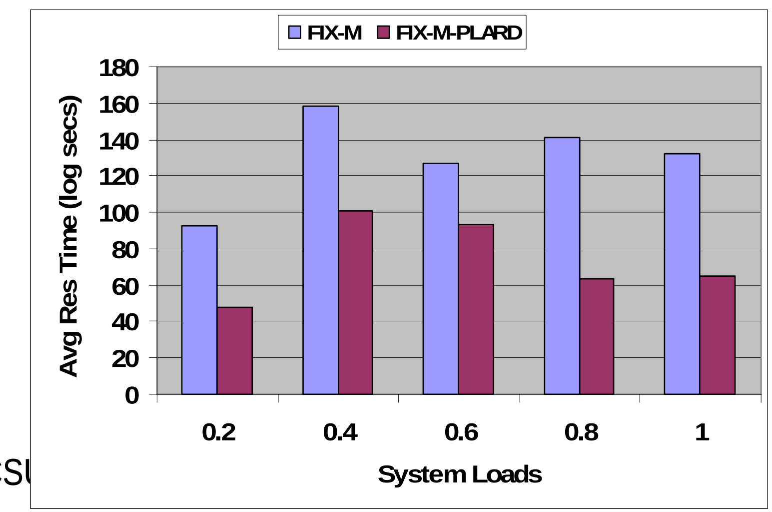
Small Partition





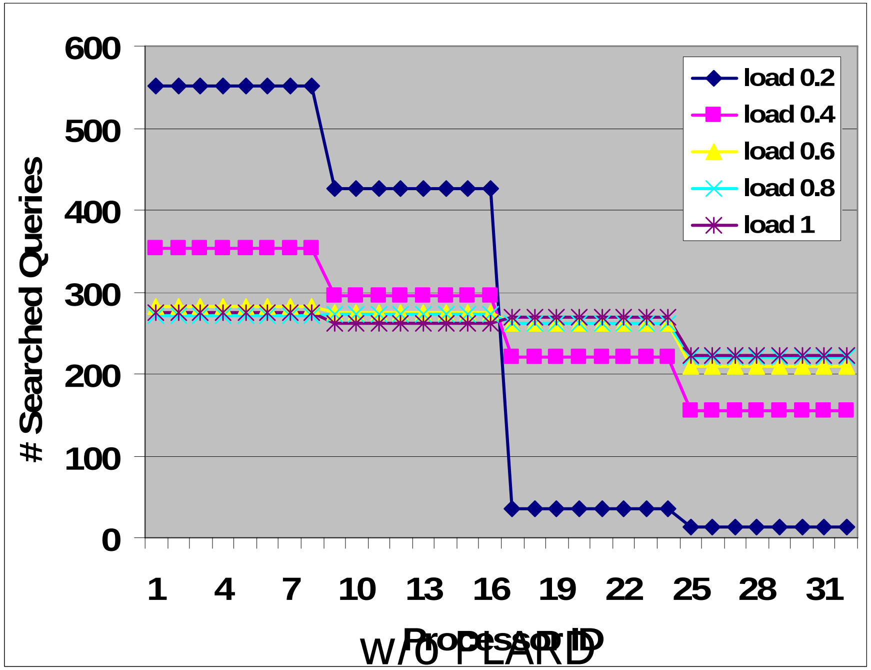
Medium Partition



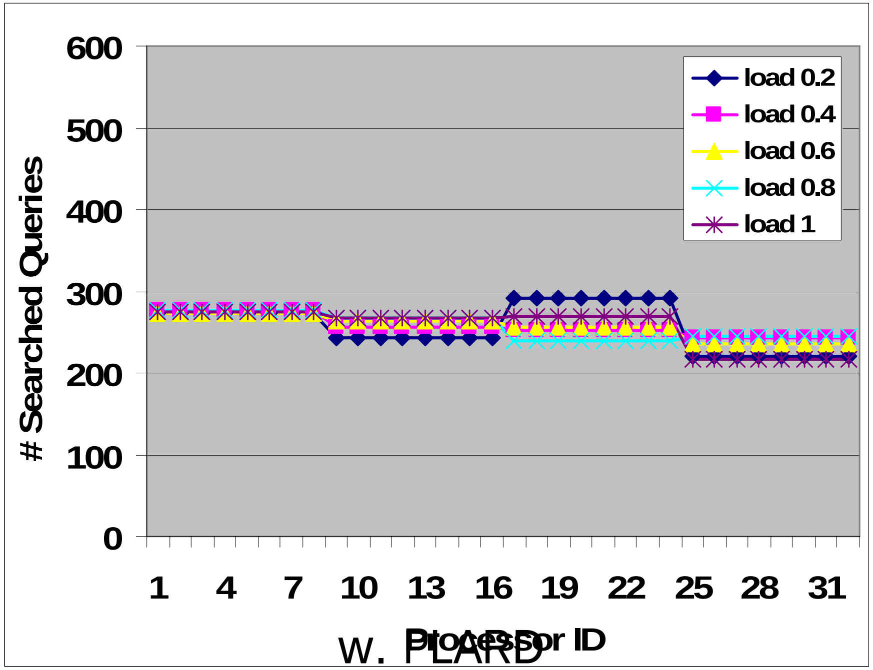


PLARD Impacts (cont.)

- Count # of searched queries on each processor
- PLARD results in more balanced loads across processors



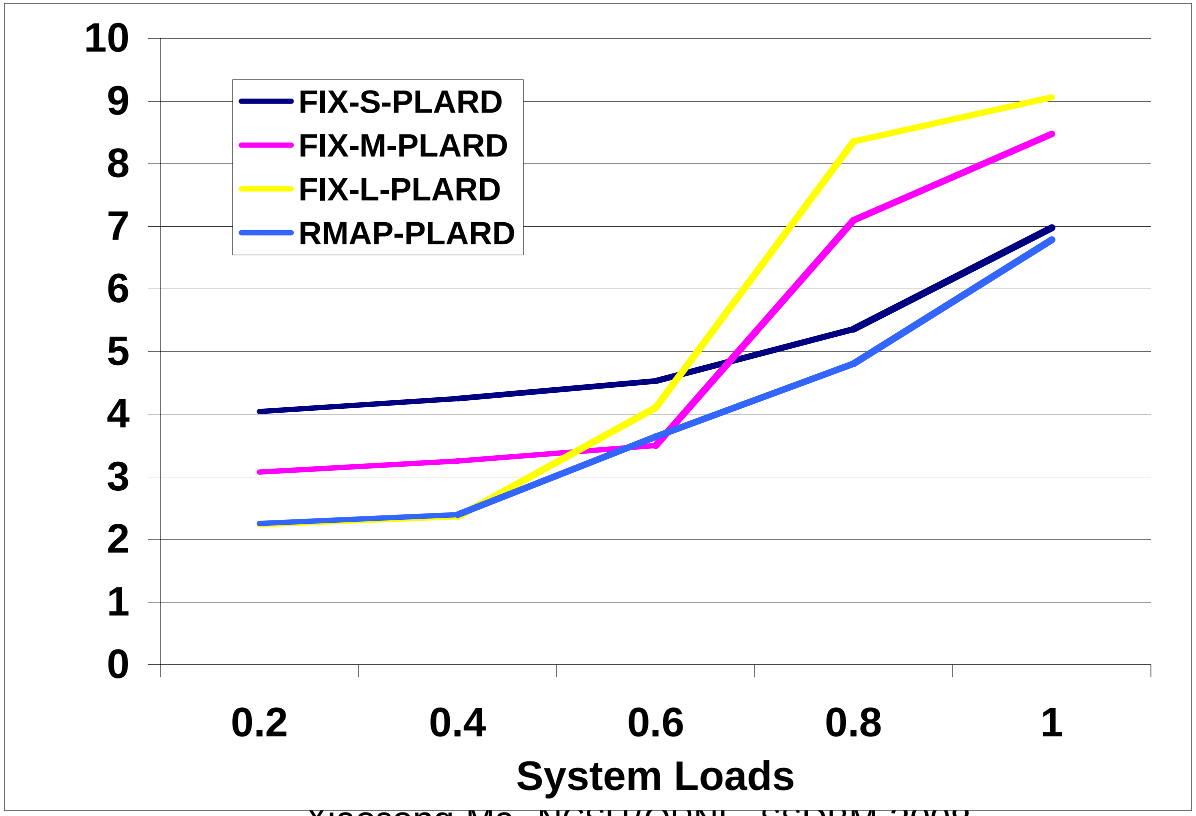
w/o PLARD



w. PLARD

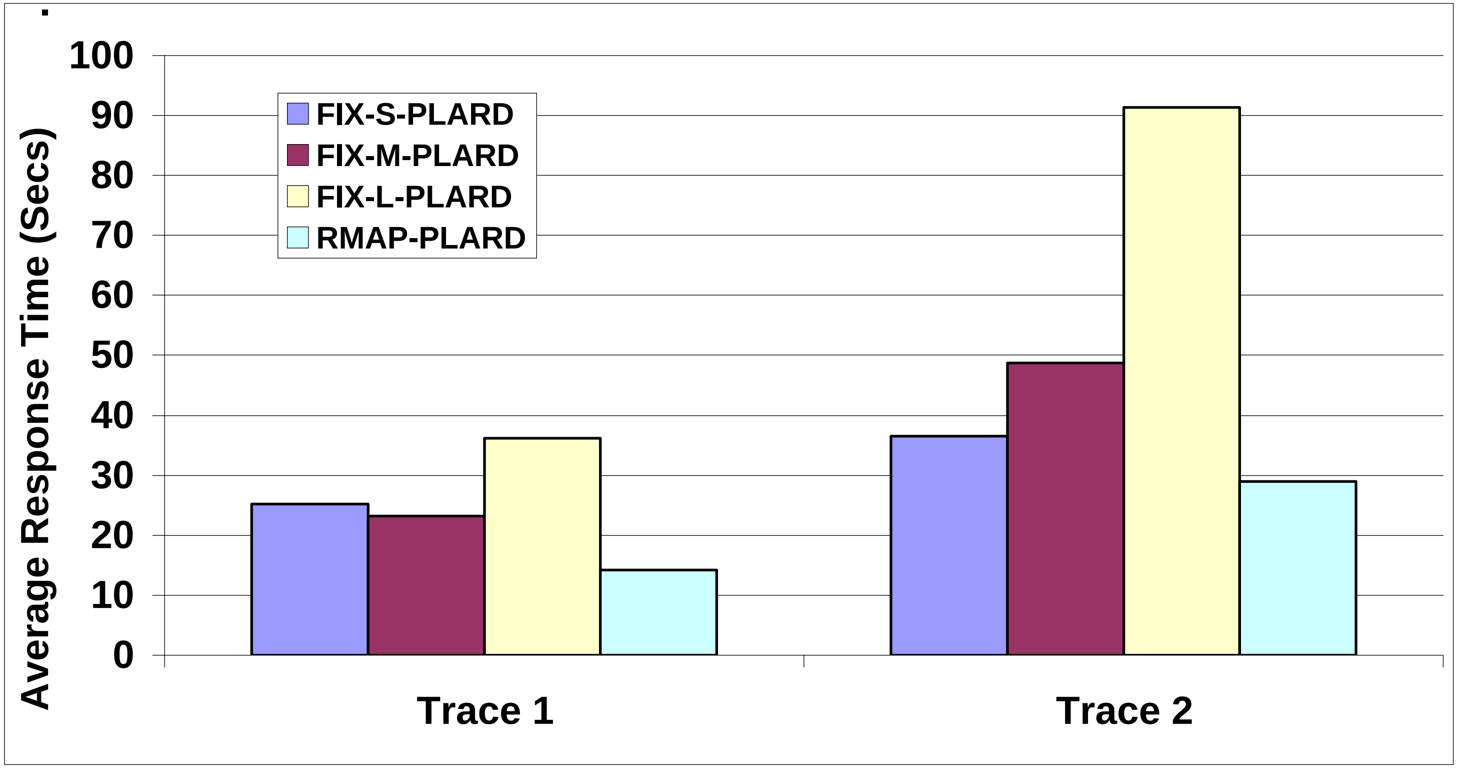
Adaptive to Fixed Arrival Rates

- Static policies work well for certain workload
- RMAP wins across the board



Adaptive to Mixed Arrival Rates

- Two traces with mixed arrival rates
  - Trace 1: 0.2 + 0.4 + 0.6 + 0.8
  - Trace 2: 0.2 + 0.8 + 0.4 + 1.0



Road Map

- Introduction
- Background
- Scheduling design
- Experiment results
- Conclusions

Conclusions

- Scientific web service request scheduling not well studied
  - "Moldable jobs" realized
- Two-level adaptive scheduling framework
  - RMAP: parallel efficiency aware
  - PLARD: data locality aware
  - Combined adaptive policy autonomically adapts to system loads and query patterns

References

- [Dandamudi99] S. Dandamudi and H. Yu. Performance of adaptive space sharing processor allocation policies for distributed-memory multicomputers. *JPDC*, 58(1), 1999.
- [Gardner06], M. Gardner, W. Feng, J. Archuleta, H. Lin, and X. Ma, Parallel Genomic Sequence-Searching on an Ad-Hoc Grid: Experiences, Lessons Learned, and Implications. *Supercomputing*, 2006
- [Lin05], H. Lin, X. Ma, P. Chandramohan, A. Geist, and N. Samatova, Efficient Data Access for Parallel BLAST, *IPDPS*, 2005.
- [Lin08], H. Lin, P. Balaji, R. Pool, C. Sosa, X. Ma, and W. Feng, Massively Parallel Genomic Sequence Search on the BlueGene/P Architecture. To appear, *Supercomputing*, 2008.
- [Pai98] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. *ASPLOS-VIII*, 1998.

Thank You

- Questions?