Decentralized Dynamic Scheduling across Heterogeneous Multi-core Desktop Grids

Jaehwan Lee, Pete Keleher, Alan Sussman

Department of Computer Science
University of Maryland

Multi-core is not enough

- Multi-core CPU is the current trend of desktop computing
- Not easy to exploit Multi-core in a ***single*** machine for high throughput computing
  - “*Multicore is Bad news for Supercomputers*”, S. Moore, IEEE Spectrum, 2008
- We have proposed ***decentralized*** solution for ***initial job placement*** for Multi-core Grids, but..

Dynamic Re-scheduling can surely improve performance even more ...

Motivation and Challenges

- Why is dynamic scheduling needed?
  - **Stale** load information
  - **Unpredictable** job completion times
  - **Probabilistic** initial job assignment

- **Challenges** for ***decentralized*** dynamic scheduling for ***multi-core*** grids
  - ***Multiple*** resource requirements
  - ***Decentralized*** algorithm needed
  - No ***Job starvation*** allowed

Our Contribution

- New ***Decentralized*** Dynamic Scheduling Schemes for ***Multi-core*** Grids
  - Intra-node scheduling
  - Inter-node scheduling
  - Aggressive job migration via Queue Balancing

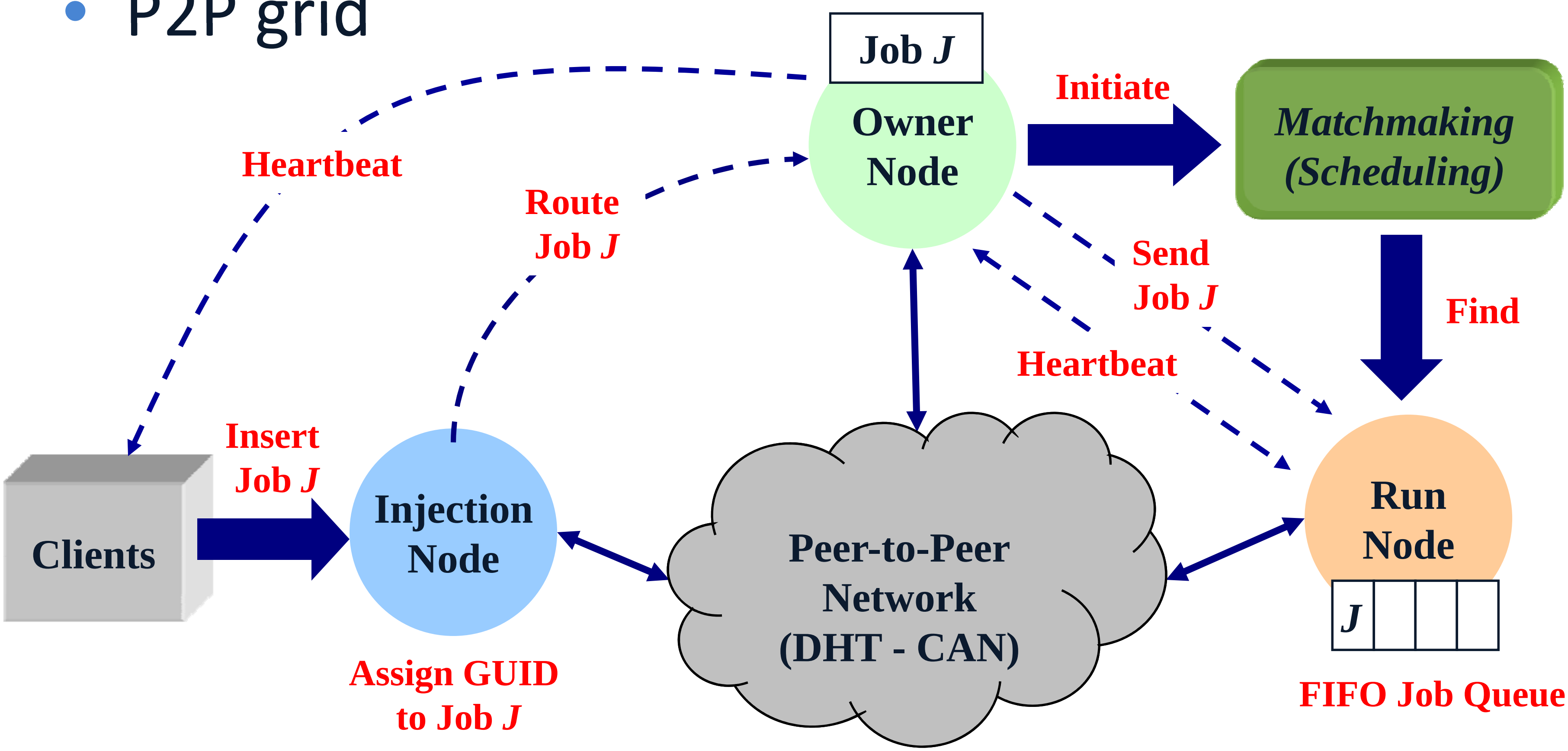
- Experimental Results via extensive simulation
  - Performance better than static scheduling
  - Competitive with an online centralized scheduler

Outline

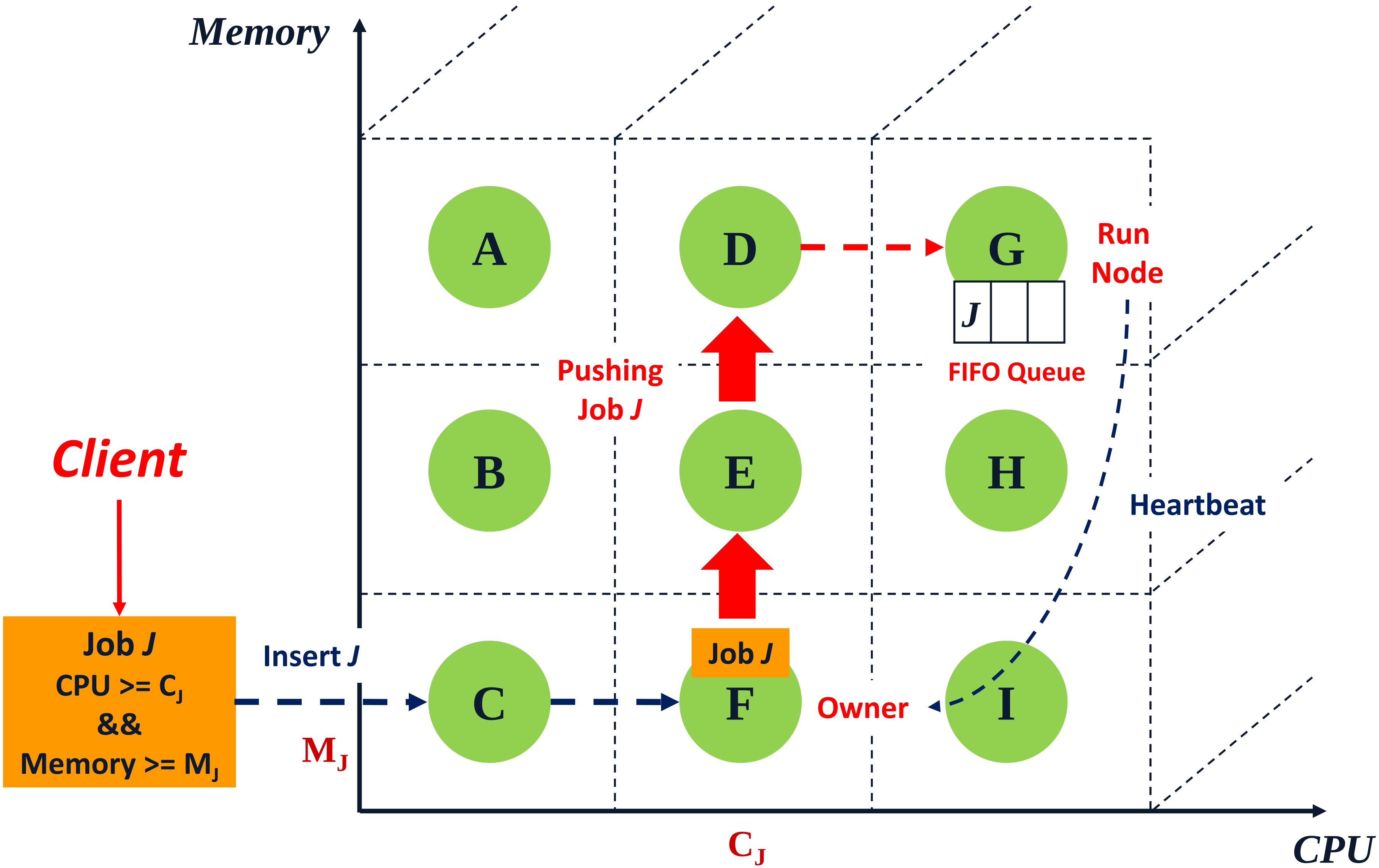
- **Background**
- Related work
- Our approach
- Experimental Results
- Conclusion & Future Work

Overall System Architecture

- P2P grid



Matchmaking Mechanism in CAN

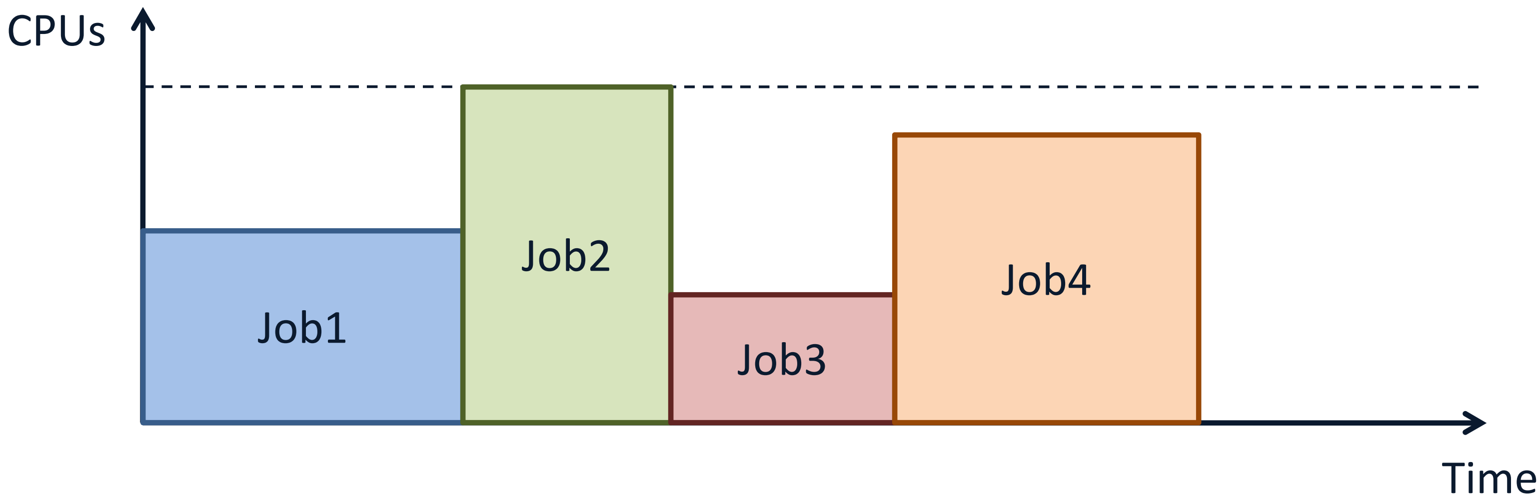


Outline

- Background
- **Related work**
- Our approach
- Experimental Results
- Conclusion & Future Work

Backfilling

- Basic Concept



- Features
  - **Job running time** must be known
  - Conservative vs. EASY Backfilling
  - Inaccurate job running time estimates reduce overall performance

Approaches for *K*-resource requirements

- **Backfilling** with ***multiple*** resource requirements (Leinberger:SC'99)
  - Backfilling in a **single** machine
  - Heuristic approaches
  - Assumption : Job Running times are known

- **Job migration** to balance K-resources between nodes (Leinberger:HCW'00)
  - Reduce local load imbalance by exchanging jobs, but does not consider overall system loads
  - No backfilling scheme
  - Assumption : near-homogeneous environment

Outline

- Background
- Related work
- **Our approach**
- Experimental Results
- Conclusion & Future Work

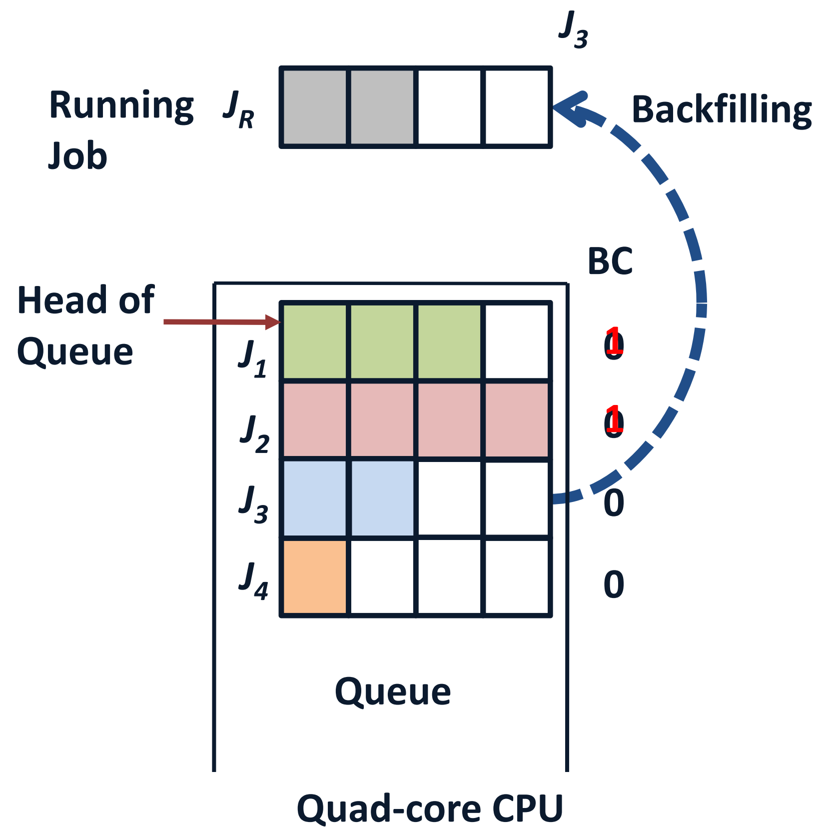
Dynamic Scheduling

- After Initial Job assignment, but before the job starts running, dynamic scheduling algorithm invoked ***Periodically***

- Costs for dynamic scheduling
  - Job Migration Cost
    - None : For intra-node scheduling
    - Minimal : For inter-node scheduling & Queue balancing
  - CPU cost : None
    - No preemptive scheduling : Once a job starts running, it won't be stopped due to dynamic scheduling.

Intra-Node Scheduling

- Extension of Backfilling with multiple resource requirements
- ***Backfilling Counter*** **(BC)**
  - Initial value : 0
  - Counts number of other jobs that have bypassed the job
  - Only a job whose BC is equal to or greater than maximum BC of jobs in the queue can be backfilled
  - No job starvation

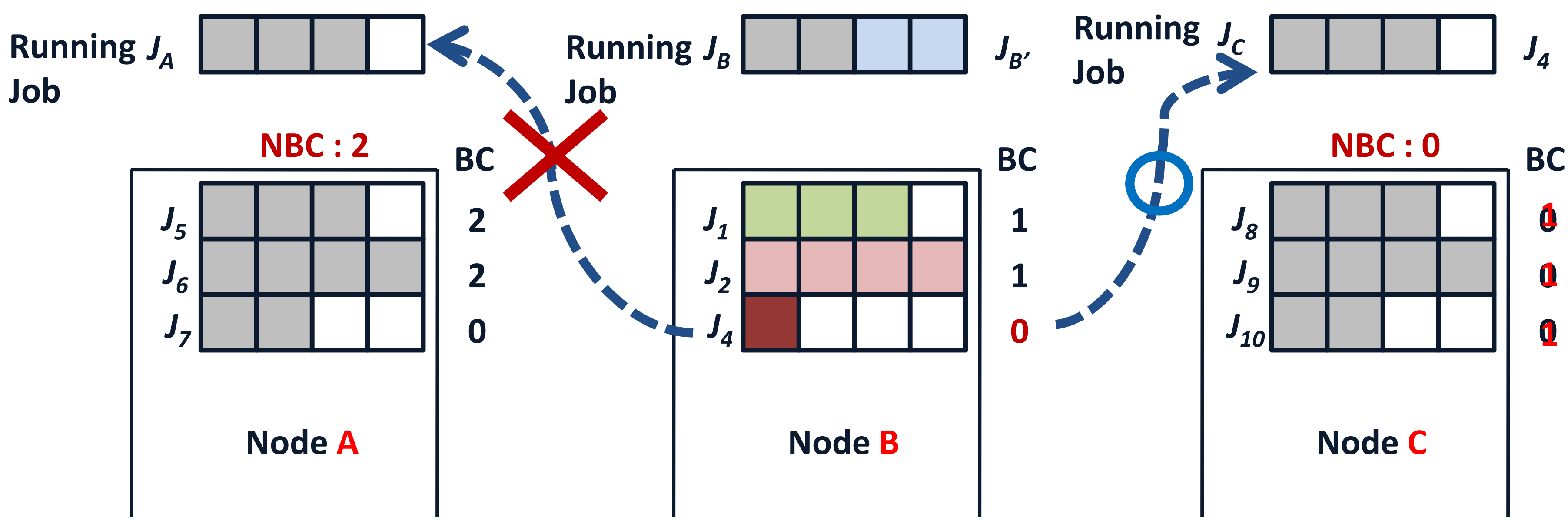


Which job should be backfilled?

- If multiple jobs can be backfilled,
  - Backfill Balanced (**BB**) (Leinberger:SC'99) algorithm
  - Choose the job with ***minimum objective function***(= **BM** x **FM**)
- **Balance Measure (BM)**
  - $\textbf{BM} = \dfrac{\text{Maximum Utilization}}{\text{Average Utilization}}$
  - ***Minimize uneven usage*** across multiple resources
- **Fullness Measure (FM)**
  - **FM** = 1 – Average Utilization
  - ***Maximize average utilization***

Inter-node Scheduling

- Extension of **Intra-node scheduling** across nodes
- **Node Backfilling Counter** (**NBC**)
  - **Maximum** BC of jobs in the node's waiting queue
  - Only jobs whose **BC** is equal to or greater than **NBC** of the target node can be migrated
  - No ***job starvation***



Inter-node Scheduling – PUSH vs. PULL

- **PUSH**
  - A job **sender** initiates the process
  - Sender tries to match every job in its queue with residual resources in its neighbors in the CAN
  - If a job can be sent to multiple nodes, pick the node with minimum objective function, and prefer a node with the fastest CPU

$$f_{Inter-PUSH} = BM \cdot FM \cdot \frac{1}{CPU_{SPEED}}$$

- **PULL**
  - A job **receiver** initiates the process
  - Receiver sends a ***PULL-Request*** message to the potential sender (the one with maximum current queue length)
  - Potential sender checks whether it has a job that can be backfilled, and the job satisfies ***BC*** condition
  - If multiple jobs can be sent, choose the job with minimum objective function (= ***BM*** x ***FM***)
  - If no job can be found, send a ***PULL-Reject*** message to receiver
  - The receiver looks for another potential sender among neighbors, if gets a ***PULL-Reject*** message

Queue Balancing

- Intra-node scheduling & Inter-node scheduling look for job that can start running immediately, to use current residual resources
- Add ***Proactive*** job migration for queue (load) balancing
  - Migrated job does not have to start immediately
- Use normalized **Load** measure for a node with multiple resources (Leinberger:HCW'00)
  - For each resource, sum all job's requirements in the queue and normalize it with respect to node's resource capability
  - **Load** on a node defined as the **maximum** of those
- PUSH & PULL schemes can be used
  - Minimize **total** local loads (= sum of loads of neighbors, **TLL**)
  - Minimize **maximum** local load among neighbors (**MLL**)

Outline

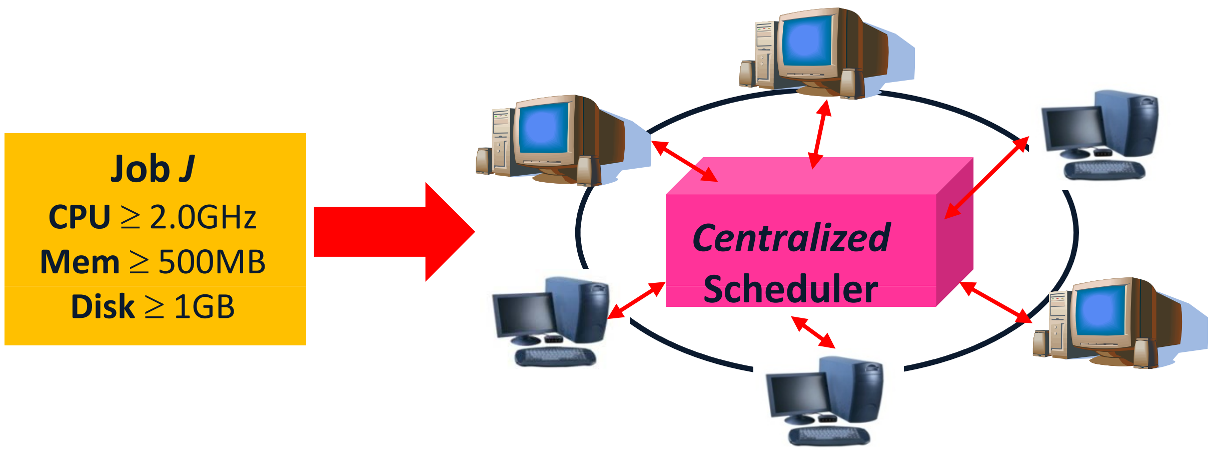
- Background
- Related work
- Our approach
- **Experimental Results**
- Conclusion & Future Work

Experimental Setup

- Event-driven Simulations
  - A set of *nodes* and *events*
    - **1000** initial nodes and **5000** job submissions
    - Jobs are submitted with average inter-arrival time **τ** (with a ***Poisson*** distribution)
    - A node has **1,2,4** or **8** cores
    - Job run times uniformly distributed between 30 and 90 minutes
  - Node Capabilities and Job Requirements
    - CPU, Memory, Disk and **the number of cores**
    - Job requirement for a resource can be omitted (Don't care)
      - **Job Constraint Ratio** : The probability that each resource type for a job is specified
  - ***Steady state*** experiments

Comparison Models

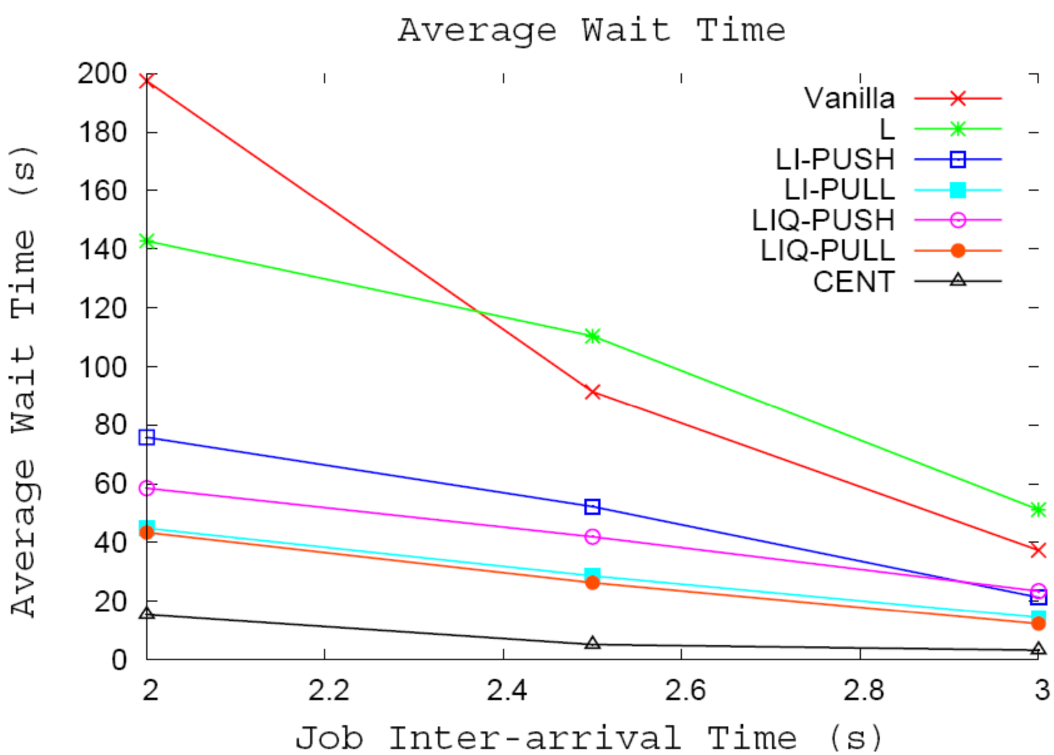
- ***Centralized*** Scheduler (**CENT**)
  - *Online* and *global* scheduling mechanism with a single wait queue
  - Not feasible in a complete implementation of P2P system



- Tested combinations of our schemes
  - **Vanilla** : No dynamic scheduling (Static Scheduling only)
  - **L** : Intra-node scheduling only
  - **LI** : **L** + Inter-node scheduling
  - **LIQ** : **LI** + Queue balancing
  - **LI(Q)-PUSH/PULL** : LI & LIQ with PUSH/PULL options

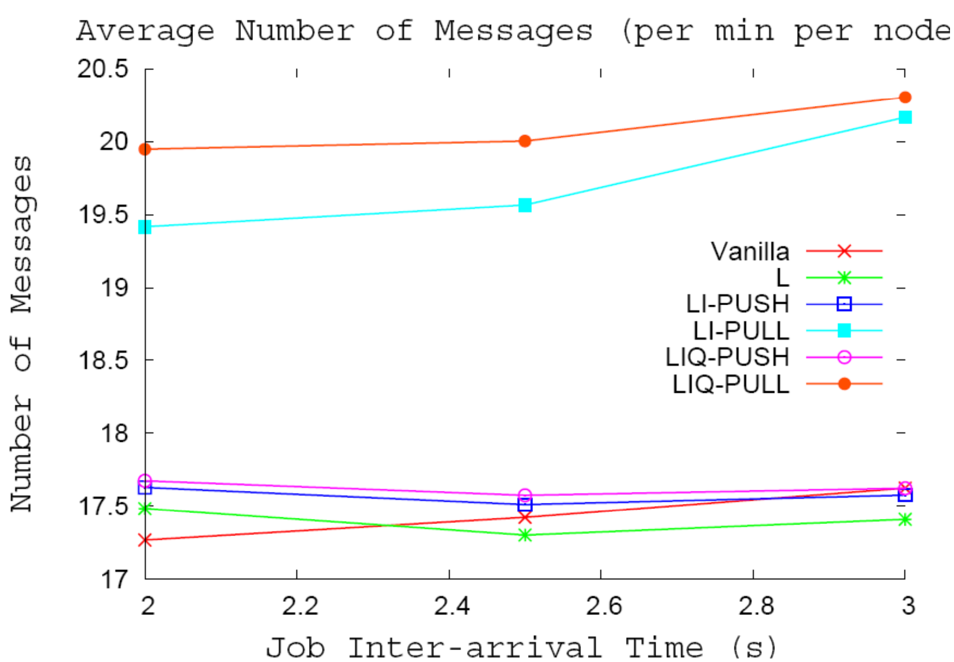
Performance varying system load

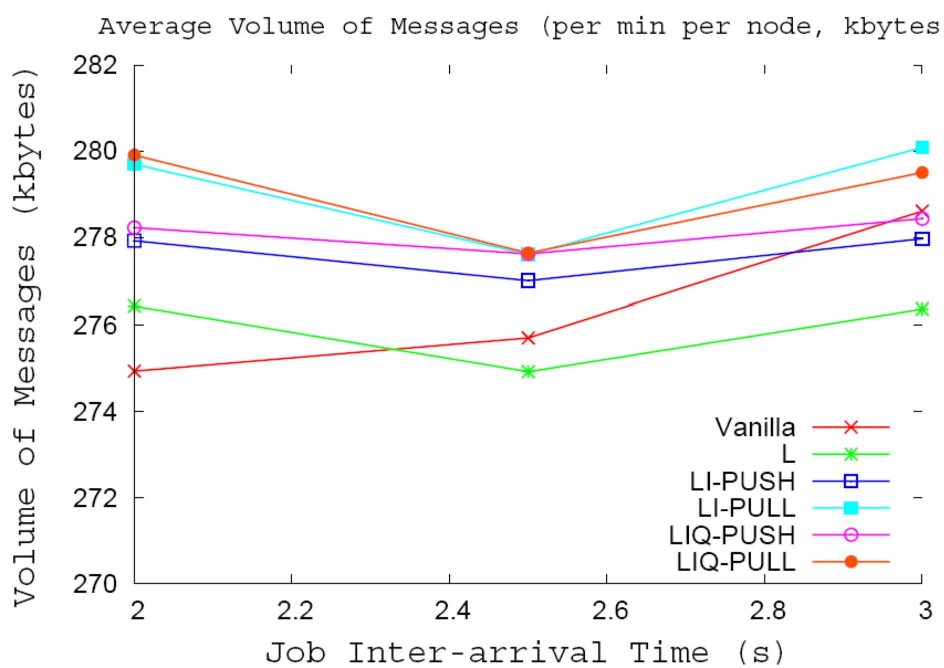
- **LIQ-PULL** > **LI-PULL** > **LIQ-PUSH** > **LI-PUSH** > **L** >= **Vanilla**
- **Inter-node scheduling** provides big improvement
- **PULL** is better than **PUSH**
  - In overloaded system, PULL is better to spread information due to ***aggressive*** trial for job migration (Demers:PODC'87)
- Intra-node scheduling cannot guarantee better performance than Vanilla
  - The **Backfilling Counter** does not ensure that other waiting jobs will not be delayed (different from conservative backfilling)



Overheads

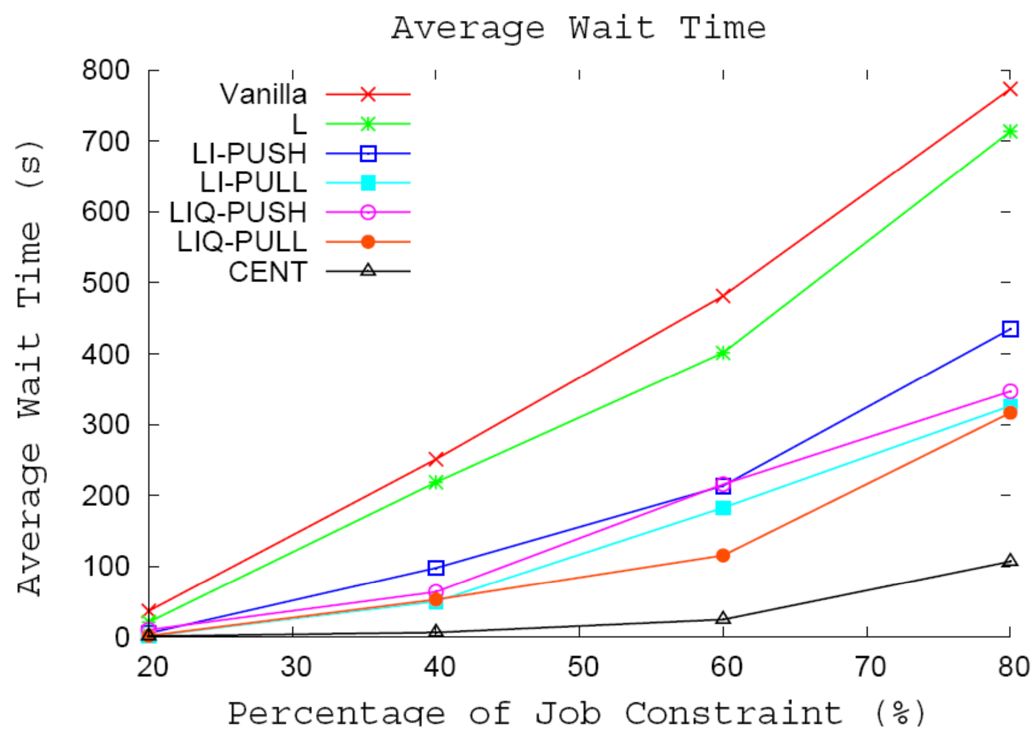
- **PULL** has ***higher*** cost than **PUSH**
  - Active search (lots of trials and rejects)
- Other schemes are similar to Vanilla
  - No significant additional overhead





Performance varying Job Constraint Ratio

- **LIQ-PULL : best**
- **LIQ == LI**
- **LIQ-PULL** is competitive to **CENT**
- For 80% Job Constraint Ratio, LIQ-PULL performance gets relatively worse
  - difficult to find a capable neighbor for job migration, because jobs are more highly constrained



Evaluation Summary

- Performance
  - **LIQ-PULL** is competitive to **CENT**
  - **Inter-node Scheduling has major impact on performance**
  - **PULL** is better than **PUSH** (more aggressive search)
  - Good performance can be achieved regardless of system load and job constraint ratio
    - it’s worthwhile to do dynamic load balancing
- Overheads
  - **PULL > PUSH** (more aggressive search)
  - Competitive to **Vanilla**

Conclusion and Future Work

- New ***decentralized*** Dynamic Scheduling for ***Multi-core*** P2P Grids
  - Extension of ***Backfilling*** (Intra-node/Inter-node)
  - ***Backfilling Counter*** : No Job Starvation
  - Proactive ***Queue Balancing***
- Performance Evaluation via simulation
  - ***Better*** than Static Scheduling
  - ***Competitive*** performance to **CENT**
  - ***Low overhead***
- Future work
  - Real grid experiments (in cooperation with Astronomy Dept.)
  - Decentralized Resource Management for ***Heterogeneous*** **Asymmetric** Multi-processors