A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid

Atsuko Takefusa, JSPS/TITECH

Henri Casanova, UCSD/SDSC

Satoshi Matsuoka, TITECH/JST

Francine Berman, UCSD/SDSC

http://ninf.is.titech.ac.jp/bricks/

The Computational Grid

- A promising platform for the deployment of HPC applications
- A crucial issue is Scheduling
  - Most scheduling works aim at improving execution time of a single application

    E.g., AppLeS, APST, AMWAT, MW, performance surface, stochastic scheduling, etc.

NES: Network-enabled Server

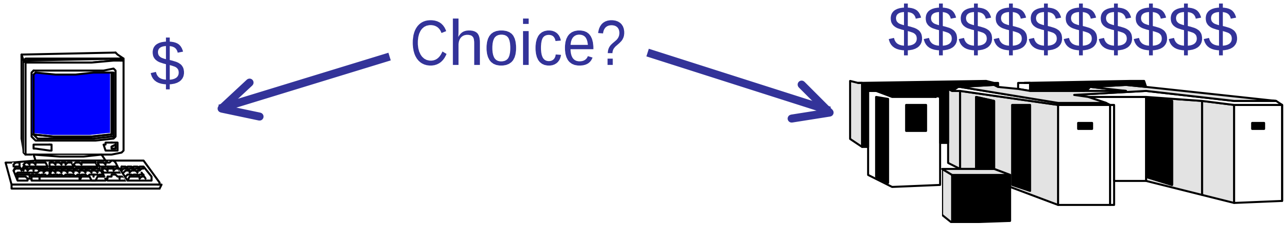
- Grid software which provides a service on the network (a.k.a. GridRPC)
  - e.g. Ninf, NetSolve, Nimrod
- Client-server architecture
- RPC-style programming model
- Many high-profile applications from science and engineering are amenable:
  - Molecular biology, genetic information, operations research

Scheduling in multi-client multi-server scenario?

Scheduling for NES

- Resource economy model (E.g. [Zhao and Karamcheti '00], [Plank '00], [Buyya '00])

  Grid currency allow owners to "charge" for usage



? No actual economical model is implemented

- Nimrod [abramson '00] presents a study of deadline-scheduling algorithm

  Users specify deadlines for the task of their apps.
  and can spend more to get tighter deadlines

Our Approach

- Our goal is to minimize
  - The overall occurrences of deadline misses
  - The resource cost

- Each request comes with a deadline requirement
- Deadline-scheduling algorithm under simple economy model
- Simulation on Bricks

  A performance evaluation system for Grid scheduling

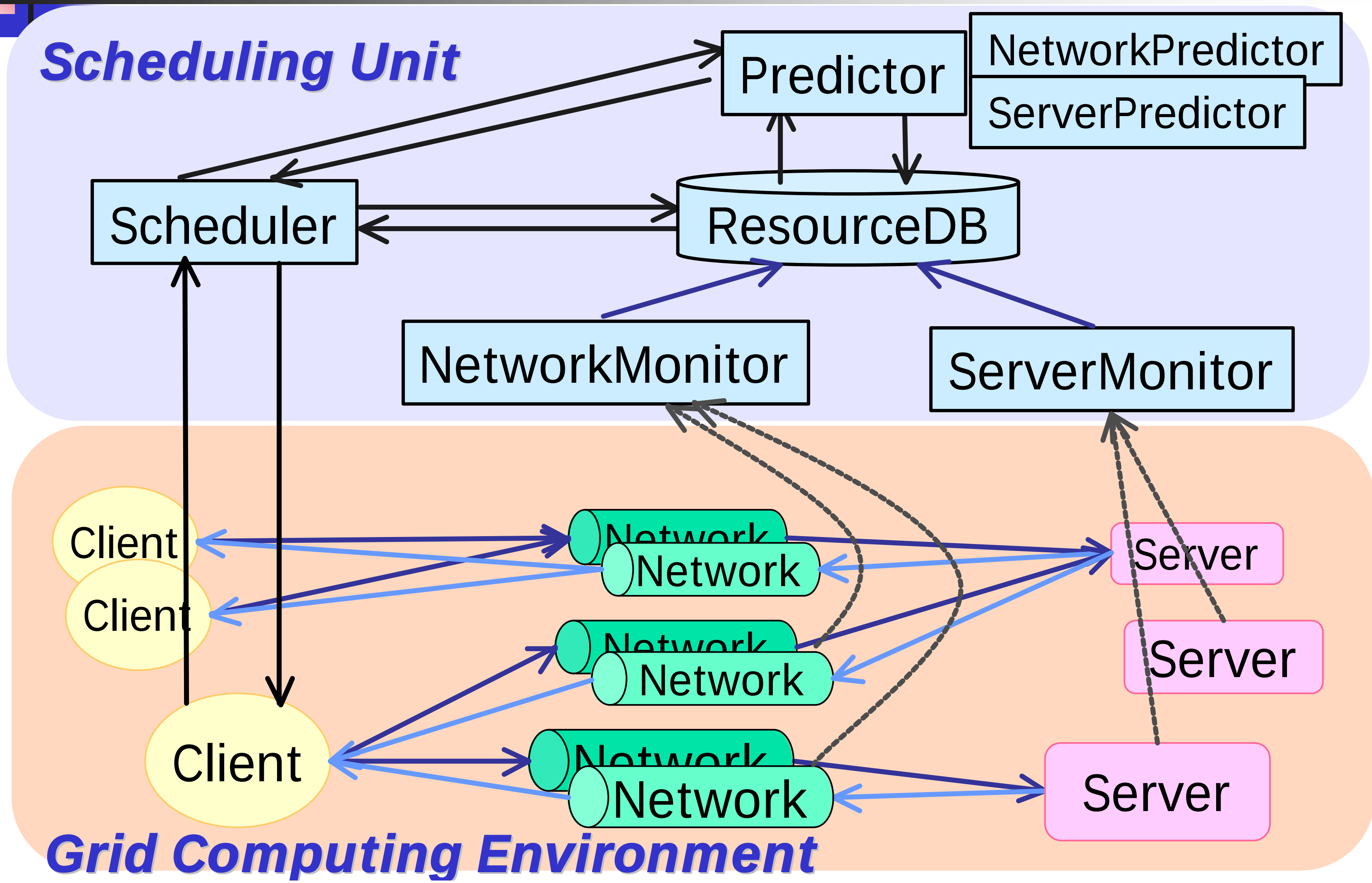
The Rest of the Talk

- Overview of Bricks and its improvement
  - More scalable and realistic simulations
- A Deadline-scheduling algorithm for multi-client/server NES systems
  - Load Correction mechanism
  - Fallback mechanism
- Experiments in multi-client multi-server scenarios with Bricks
  - Resource load, resource cost, conservatism of prediction, efficacy of our deadline-scheduling

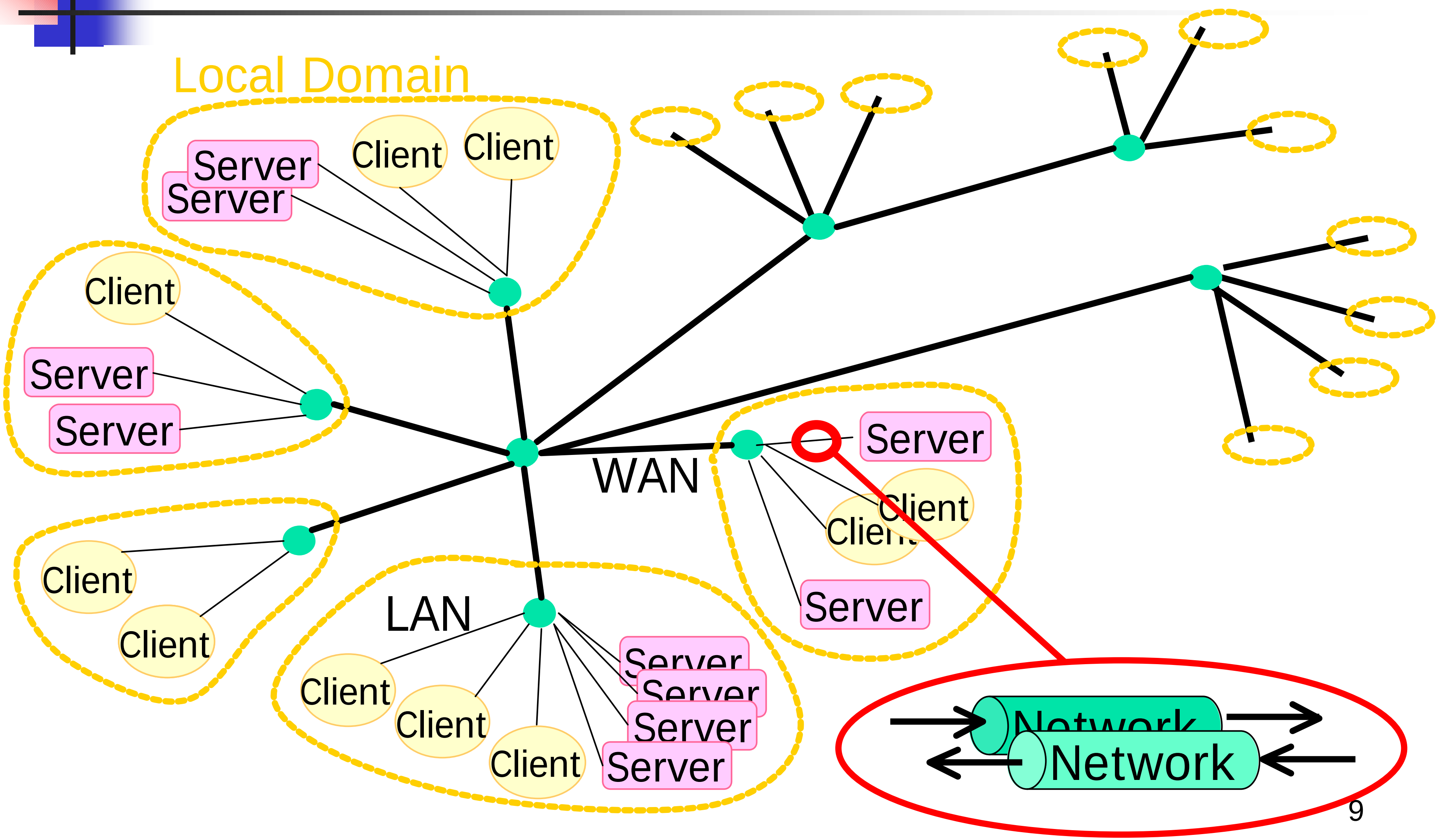
Bricks: A Grid Performance Evaluation System [HPDC '99]

- A Grid simulation framework to evaluate
  - Scheduling algorithms
  - Scheduling framework components (e.g. predictors)
- Bricks provides
  - Reproducible and controlled Grid evaluation environments
  - Flexible setups of simulation environments (Grid topology, resource model, client model)
  - Evaluation environment for external Grid components (e.g., NWS forecaster)

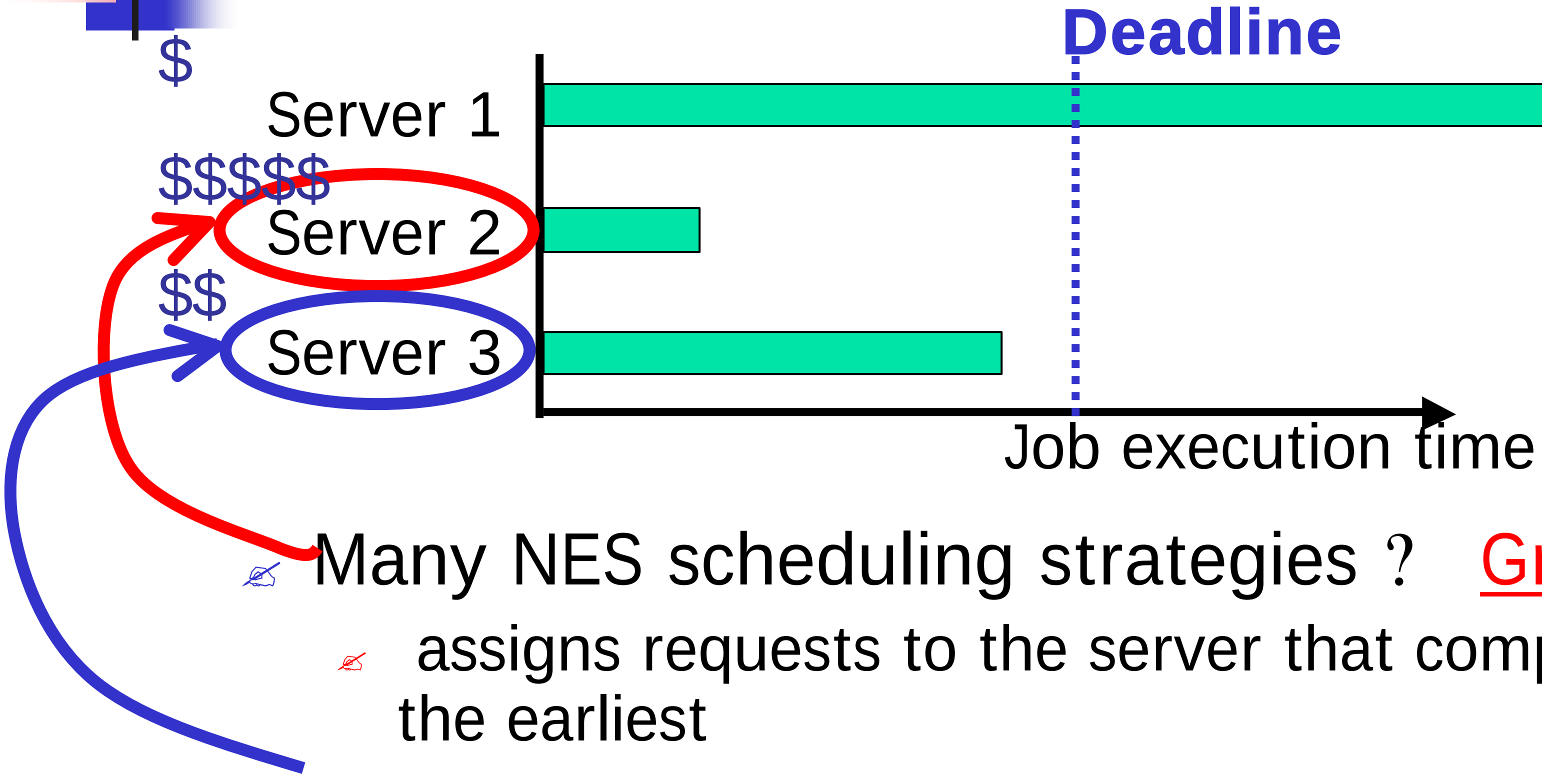
The Bricks Architecture [HPDC '99]



A Hierarchical Network Topology on the improved Bricks



Deadline-Scheduling



- Many NES scheduling strategies ? <u>Greedy</u>
  - assigns requests to the server that completes it the earliest
- <u>Deadline-scheduling:</u>
  - Aims at meeting user-supplied job deadline specifications

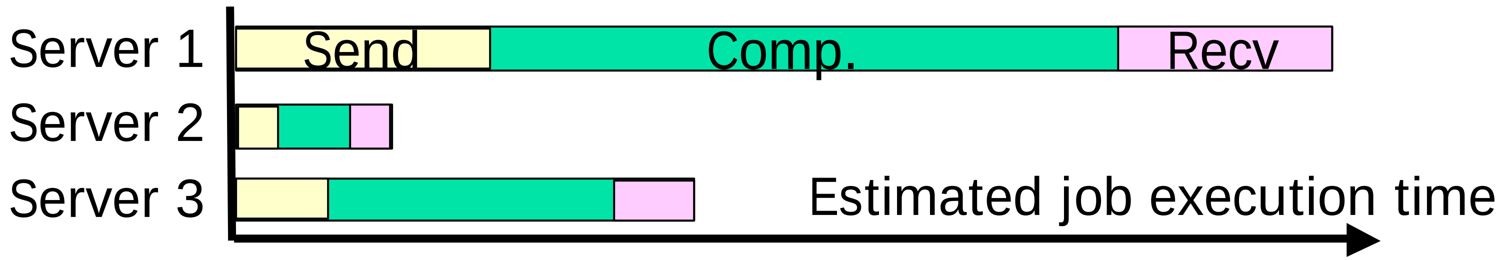
A Deadline-Scheduling Algorithm for multi-client/server NES

1. Estimate job processing time T_{si} on each server S_i :

$T_{si} = W_{send}/P_{send} + W_{recv}/P_{recv} + W_s/P_{serv}$ (0 ? $i < n$)

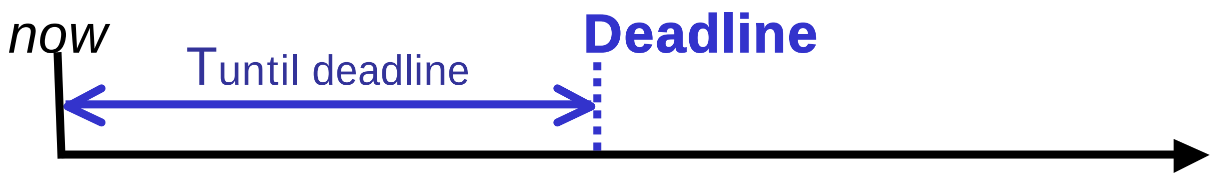
W_{send}, W_{recv}, W_s: send/recv data size, and logical comp. cost

P_{send}, P_{recv}, P_{serv}: estimated send/recv throughput, and performance



2. Compute $T_{until\ deadline}$:

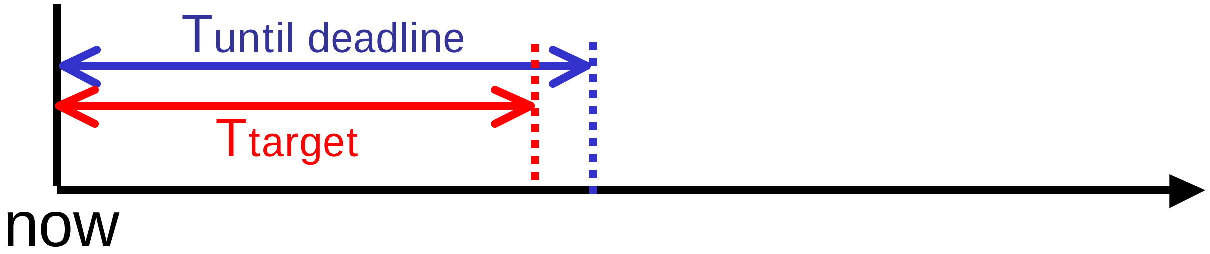
$T_{until\ deadline} = T_{deadline} - now$



A Deadline-Scheduling Algorithm (cont.)

3 Compute target processing time T_{target}:

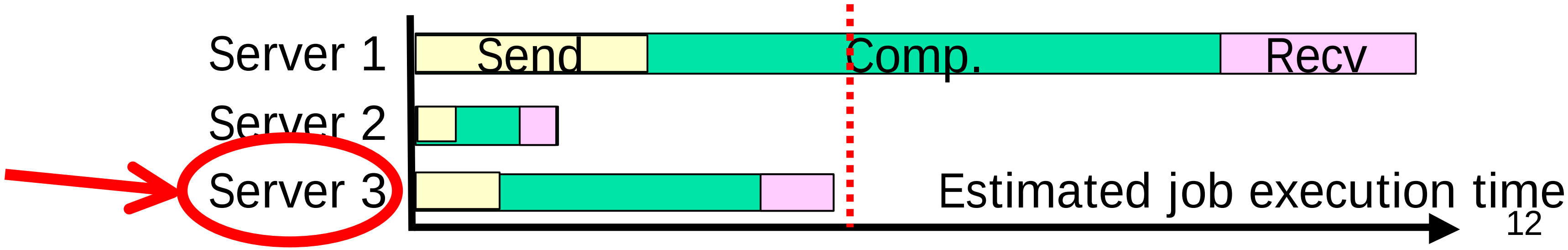
$T_{target} = T_{until\ deadline} \times Opt$ (0 < Opt ? 1)



4 Select suitable server S_i:

Conditions : *MinDiff*=Min(*Diff* $_{si}$) where *Diff* $_{si} = T_{target} - T_{si}$? 0

Otherwise Min(|*Diff*|)



Factors in Deadline-Scheduling Failures

- Accuracy of predictions is not guaranteed
- Monitoring systems do not perceive load change instantaneously
- Tasks might be out-of-order in FCFS queues

Ideas to improve schedule performance

- ✍ Scheduling decisions will result in an increase in load of scheduled nodes

? Load Correction: Use corrected load values

- ✍ Server can estimate whether it will be able to complete the task by the deadline

? Fallback: Push a scheduling functionality to server

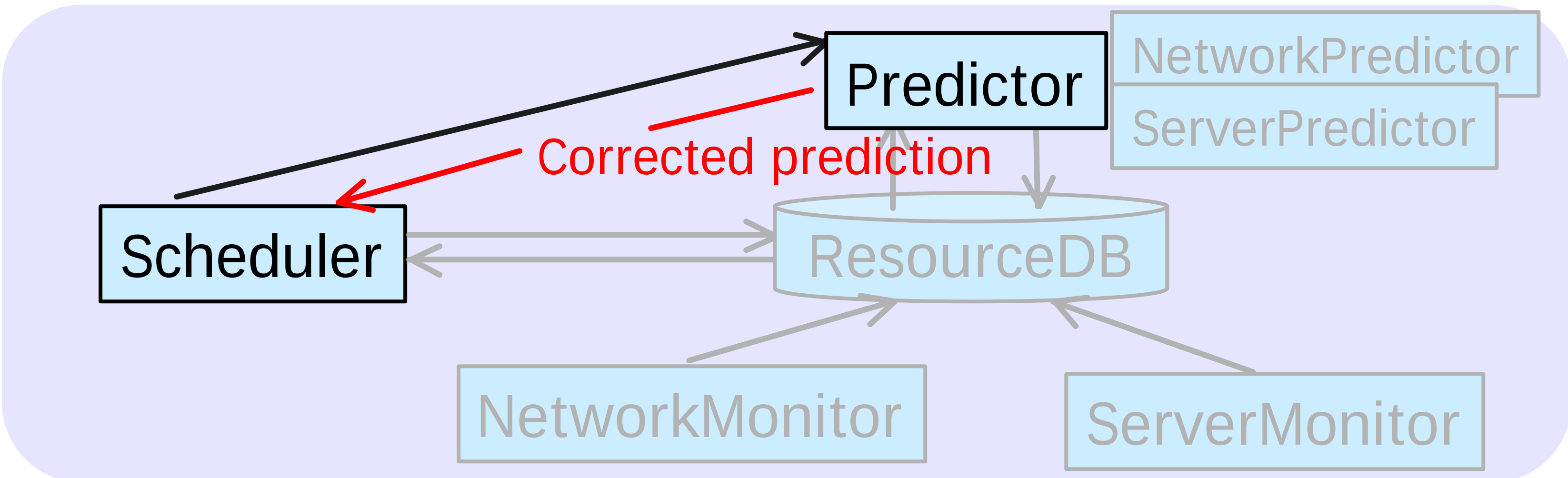
The Load Correction Mechanism

- Modify load predictions from monitoring system, $Load_{Si}$, as follows:

  $$Load_{Si\ corrected} = Load_{Si} + N_{jobs\ Si} \times pload$$

  $N_{jobs\ Si}$: the number of scheduled and unfinished jobs on the server S_i

  Pload (= 1): arbitrary value that determines the magnitude



The Fallback Mechanism

- Server can estimate whether it will be able to complete the task by the deadline
- Fallback happens when:

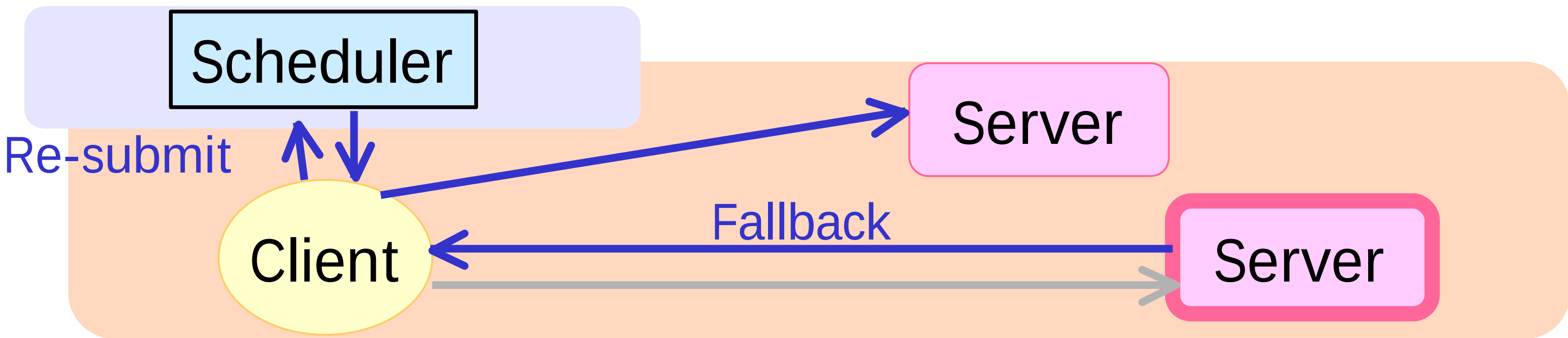
  $T_{until\ deadline} < T_{send} + ET_{exec} + ET_{recv}$ &&

  $N_{max.\ fallbacks}$? $N_{fallbacks}$

  T_{send} : Comm. duration (send)

  ET_{exec}, ET_{recv}: Estimated comm. (recv) and comp. duration

  $N_{fallbacks}$, $N_{max.\ fallbacks}$: Total/Max. number of fallbacks



Experiments

- Experiments in multi-client multi-server scenarios with Bricks
  - Resource load, resource cost, conservatism of prediction, efficacy of our deadline-scheduling
- Performance criteria:
  - Failure rate: Percentage of requests that missed their deadline
  - Resource cost: Avg. resource cost over all requests

    cost = machine performance

    E.g. select 100 Mops/s and 300 Mops/s servers

    ? Resource cost=200

Scheduling Algorithms

- Greedy: Typical NES scheduling strategy
- Deadline (Opt = 0.5, 0.6, 0.7, 0.8, 0.9)
- Load Correction (on/off)
- Fallback ($N_{max\ fallbacks}$ = 0/1/2/3/4/5)

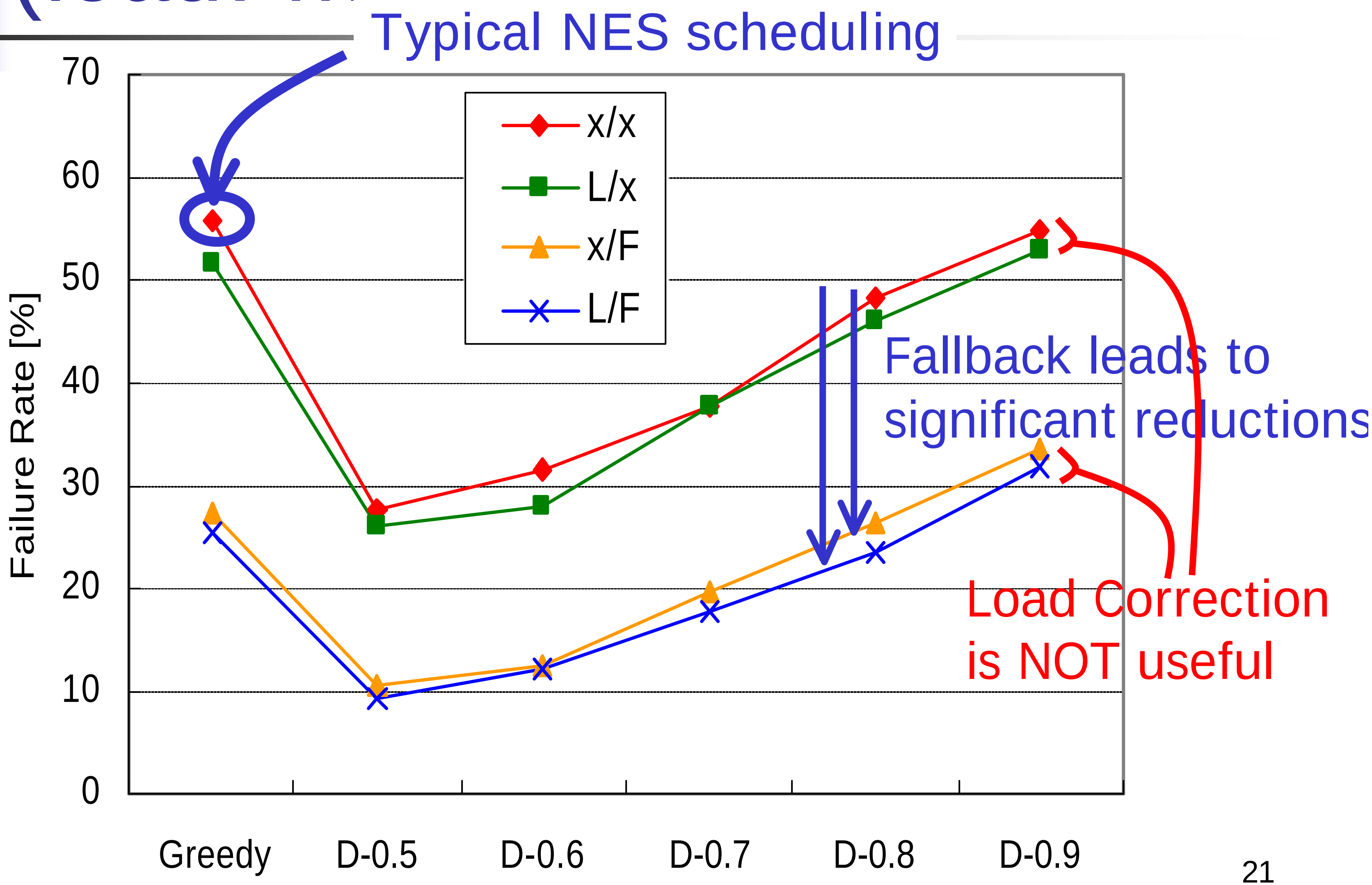
Configurations of the Bricks Simulation

- Grid Computing Environment (?75 nodes, 5 Grids)
  - # of local domain: 10, # of local domain nodes: 5-10
  - Avg. LAN bandwidth: 50-100[Mbits/s]
  - Avg. WAN bandwidth: 500-1000[Mbits/s]
  - Avg. server performance: 100-500[Mops/s]
  - Avg. server Load: 0.1
- Characteristics of client jobs
  - Send/recv data size: 100-5000[Mbits]
  - # of instructions: 1.5-1080[Gops]
  - Avg. intervals of invoking:
    60(high load), 90(medium load), 120(low load) [min]

Simulation Environment

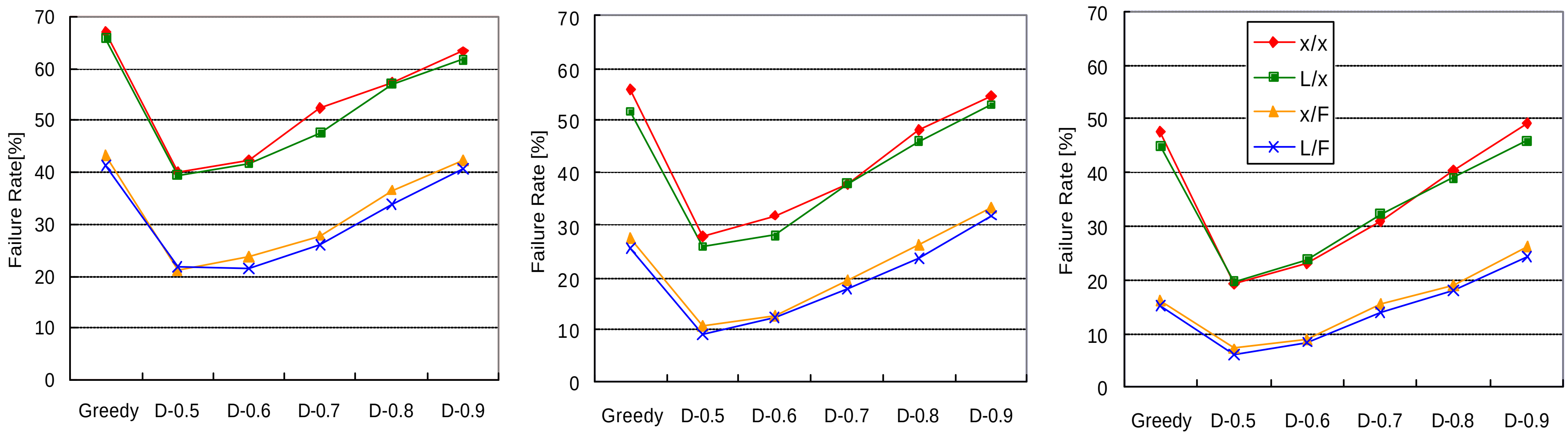
- The Presto II cluster:
  128PEs at Matsuoka Lab.,
  Tokyo Institute of Technology.
  - Dual Pentium III 800MHz
  - Memory: 640MB
  - Network: 100Base/TX
- Use APST[Casanova '00] to deploy Bricks simulations
- 24 hour simulation x 2,500 runs
  (1 sim. takes 30-60 [min]
  with Sun JVM 1.3.0+HotSpot)

Comparison of Failure Rates (load: medium)



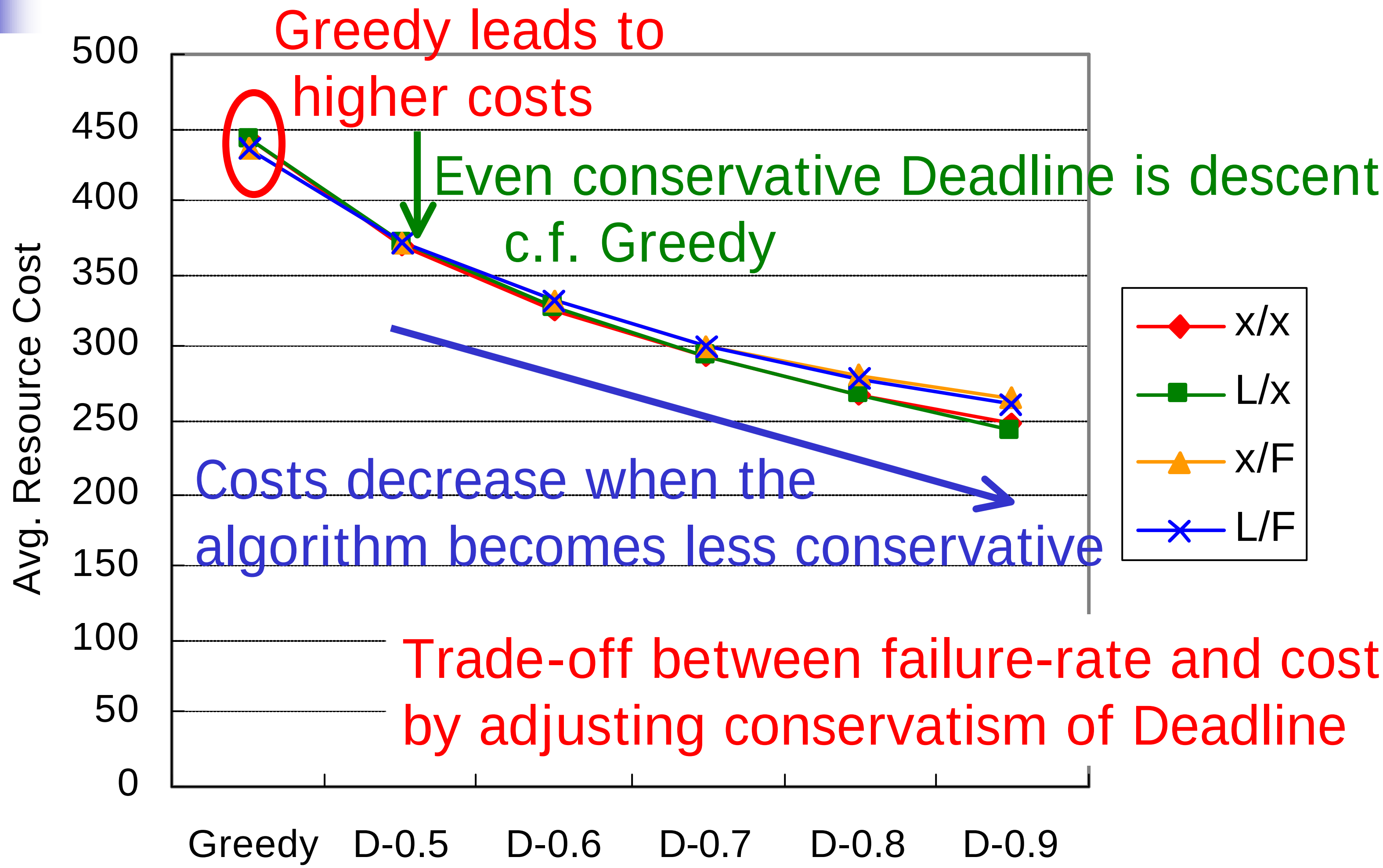
Comparison of Failure Rates (Load: high, medium, low)

High Medium Low

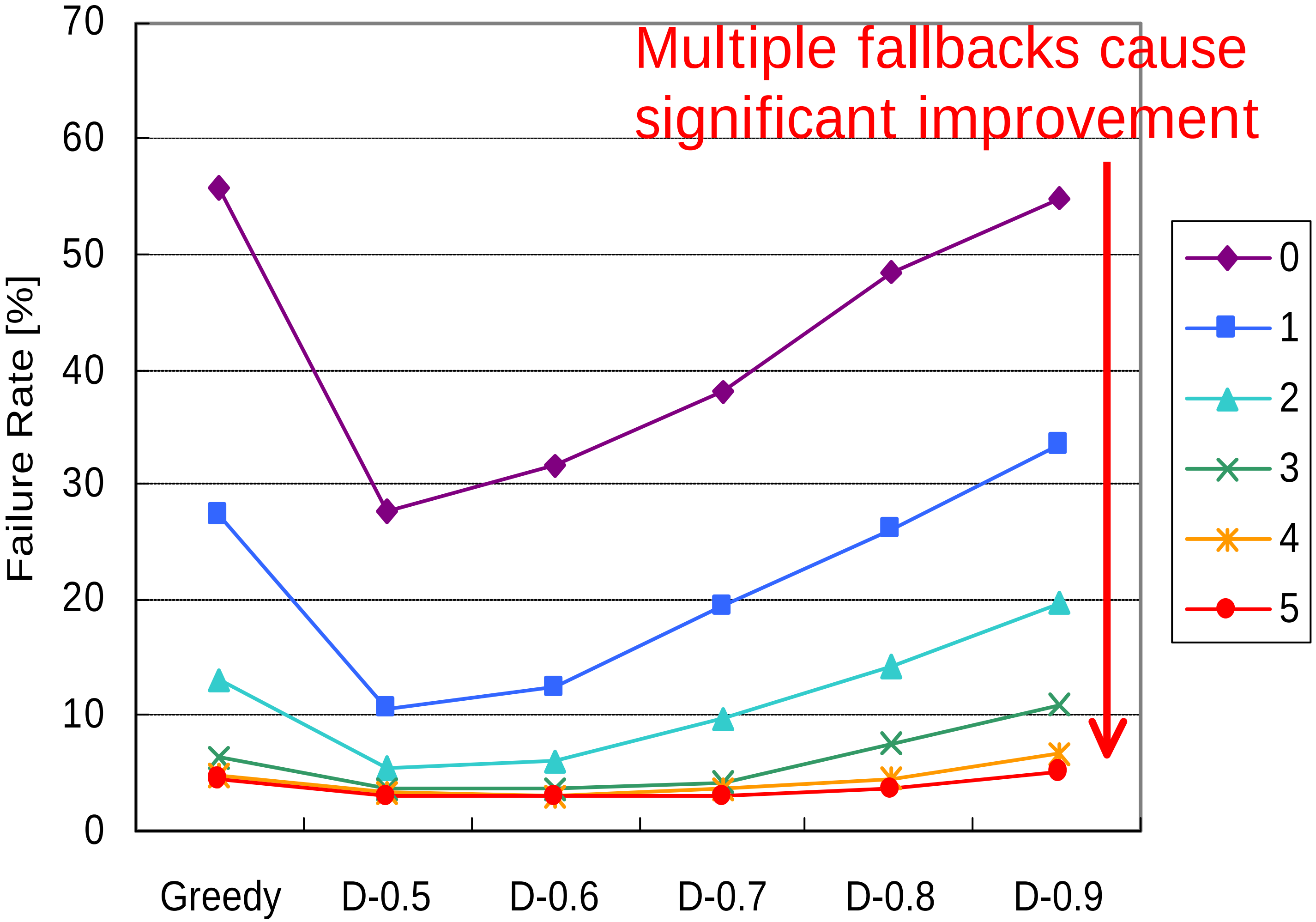


- “Low” load leads to improved failure rates
- All show similar characteristics

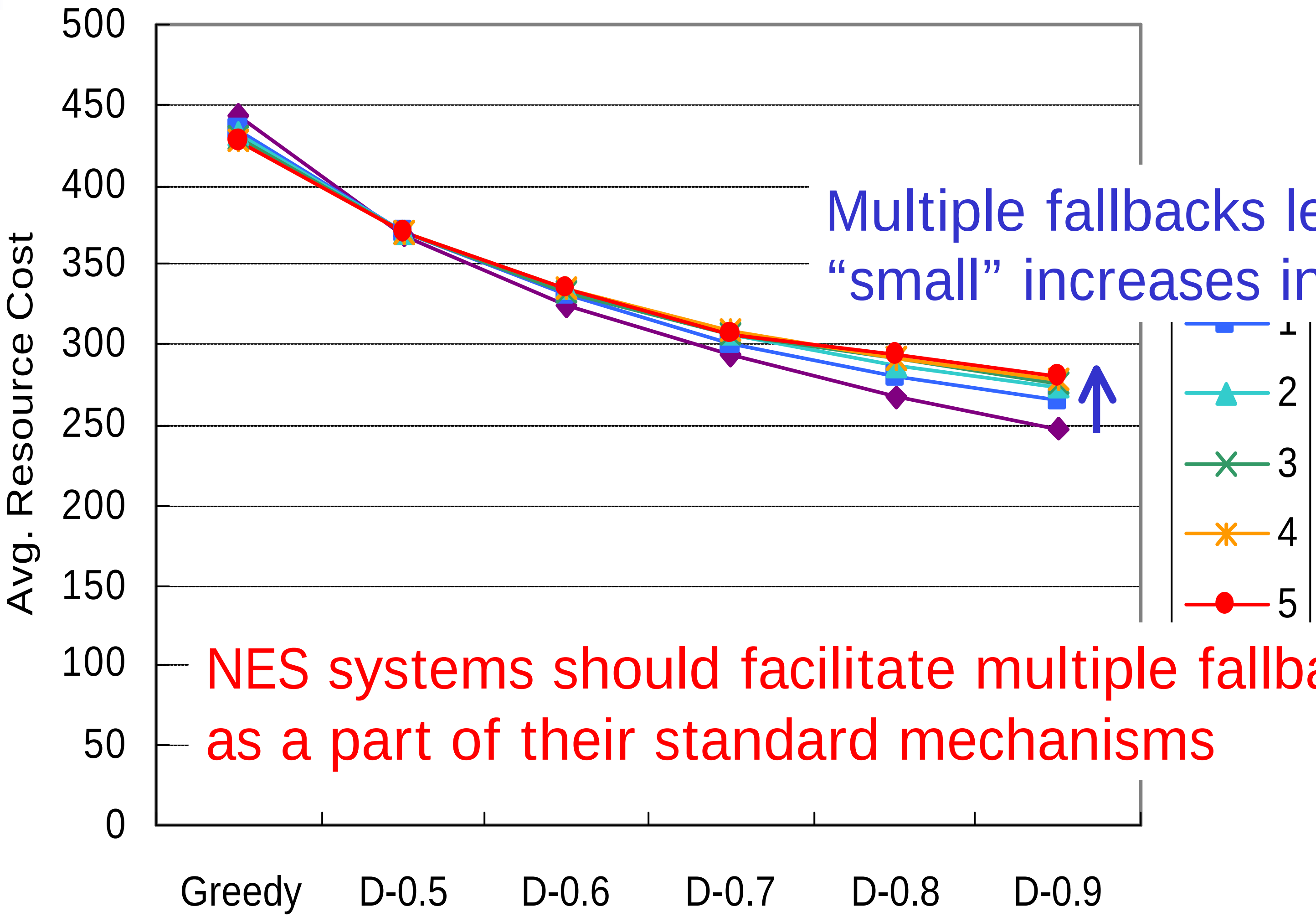
Comparison of Resource Costs



Comparison of Failure Rates (x/F, $N_{max.\ fallbacks}$ = 0-5)



Comparison of Resource Costs (x/F, $N_{max.\ fallbacks}$ = 0-5)



Multiple fallbacks lead to "small" increases in costs

NES systems should facilitate multiple fallbacks as a part of their standard mechanisms

Related Work

- Economy model:
  - Nimrod [abramson '00]
    - Uses a self-scheduler
    - Targets parameter sweep apps. from a single user
- Grid performance evaluation systems:
  - MicroGrid [Song '00]
    - Emulates a virtual Globus Grid on an actual cluster
    - Not appropriate for large numbers of experiments
  - Simgrid [Casanova '01]
    - A trace-based discrete event simulator
    - Provides primitives for simulation of application scheduling
    - Lacks the network-modeling feature Bricks provides

Conclusions

- Proposed a deadline-scheduling algorithm for multi-client/server NES systems, and Load Correction and Fallback mechanisms
- Investigated performance in multi-client multi-server scenarios with the improved Bricks
- The experiments showed
  - It is possible to make a trade-off between failure-rate and resource cost by adjusting conservatism
  - Load Correction may not be useful
  - Future NES systems should use deadline-scheduling with multiple fallbacks

Future Work

- Make Bricks support more sophisticated economy models
- Investigate their feasibility and improve our deadline-scheduling algorithms
- Implement the deadline-scheduling algorithm within actual NES systems

  (starting with Ninf: http://ninf.apgrid.org/)