The Dynamic Grid

Jon Weissman
University of Minnesota
Department of Computer Science
National E-Science Center

About Me

- Associate Professor of CS
- Strong sense of loyalty
  - working in Grid-like systems since 1995
- Original member of Legion group
  - great ideas, too radical, revolutionary
- Our philosophy
  - live within the ecosystem

Origins of the title

A) Thilo asked for an abstract/title QUICKLY?

B) Sounds like a cool term – why not?

C) Upon reflection, our research seems to be captured by this concept?

D) All of the above

Answer:

Origins of the title

A) Thilo asked for an abstract/title QUICKLY?

B) Sounds like a cool term – why not?

C) Upon reflection, our research seems to be captured by this concept?

D) All of the above

Answer: A

Origins of the title

A) Thilo asked for an abstract/title QUICKLY?

B) Sounds like a cool term – why not?

C) Upon reflection, our research seems to be captured by this concept?

D) All of the above

Answer: C (just kidding)

Origins of the title

A) Thilo asked for an abstract/title QUICKLY?

B) Sounds like a cool term – why not?

C) Upon reflection, our research seems to be captured by this concept?

D) All of the above

Answer: D

A Definition

- Dynamic: “relating or tending toward change”
- This describes the run-time behavior of distributed systems!
- Dimensions of dynamism in distributed systems
  - resources
  - demand
  - behavior

Dynamism

- Resource dynamism
  - more than one computer is involved, so I probably don't own them all
  - likely that I don't own the connecting network
- Demand dynamism
  - distributed systems promote sharing (think: client-server)
  - demand isn't always known a-priori

Dynamism (cont'd)

- Behavior Dynamism
  - people involved: resource providers, job submitters, end-clients, sys admins, network admins
  - their actions may change with time

- All of this makes distributed systems research fun yet HARD

Grid Dynamism

- The Grid “lives” at the upper-end of the dynamism spectrum
  - resources X demand X behavior
- Dynamism is a good thing
  - flexible systems emerge
  - assume the worst
    - Larry Peterson, PI of PlanetLab “inferior railroad tracks lead to superior locomotives”
- However, it must be managed

Master Class Thoughts

- We'll have a chat about "the Dynamic Grid"
- Mixture of prior research and open problems
- Goals: expose you to some of the important issues and opportunities

Questions?

Master Class Roadmap

- Four modules
- Dynamic Resources + Behavior
- Dynamic Communication
- Dynamic Metrics
- Dynamic Architectures

Collaborators

- Abhishek Chandra, UMn
- Adam Barker, NeSC
- Students: Jinoh Kim, Darin England, Krishnaveni Budati, Rahul Trivedi, Vasumathi Sundaram
- Sponsor: National Science Foundation (NSF): CNS-0305641

Context: Environment

- RIDGE project - ridge.cs.umn.edu
  - reliable infrastructure for donation grid envs
- Live deployment on PlanetLab – planet-lab.org
  - 700 nodes spanning 335 sites and 35 countries
  - emulators and simulators
- Applications
  - BLAST
  - Traffic planning
  - Image comparison

Dynamic Resources/Behavior

- Open Distributed System
  - Condor, BOINC
  - Nodes join voluntarily and behave unpredictably

- Exploit these nodes to perform computations
  - Individual jobs
  - Host compute- and data-oriented services (BLAST)

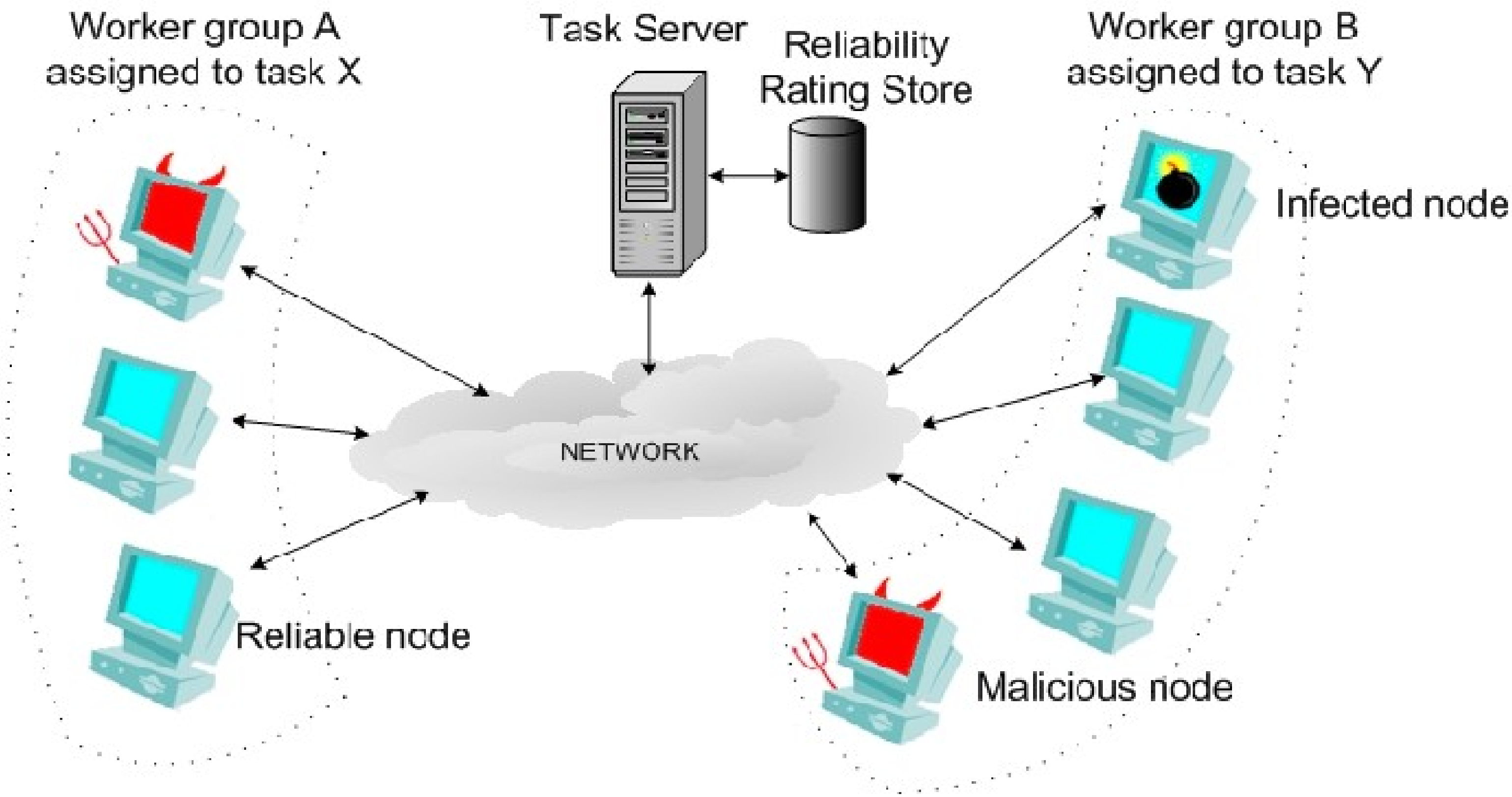
Challenge

- Nodes are unreliable – crash, hacked, churn, malicious, misconfigured, slow, etc.
  - > 1% of SETI nodes "cheated" to improve turnaround time
- How can we insure (1) correct and (2) timely results?
- Result verification techniques
  - application-specific verifiers – not general – and don't improve timeliness
- General solution: redundancy + voting

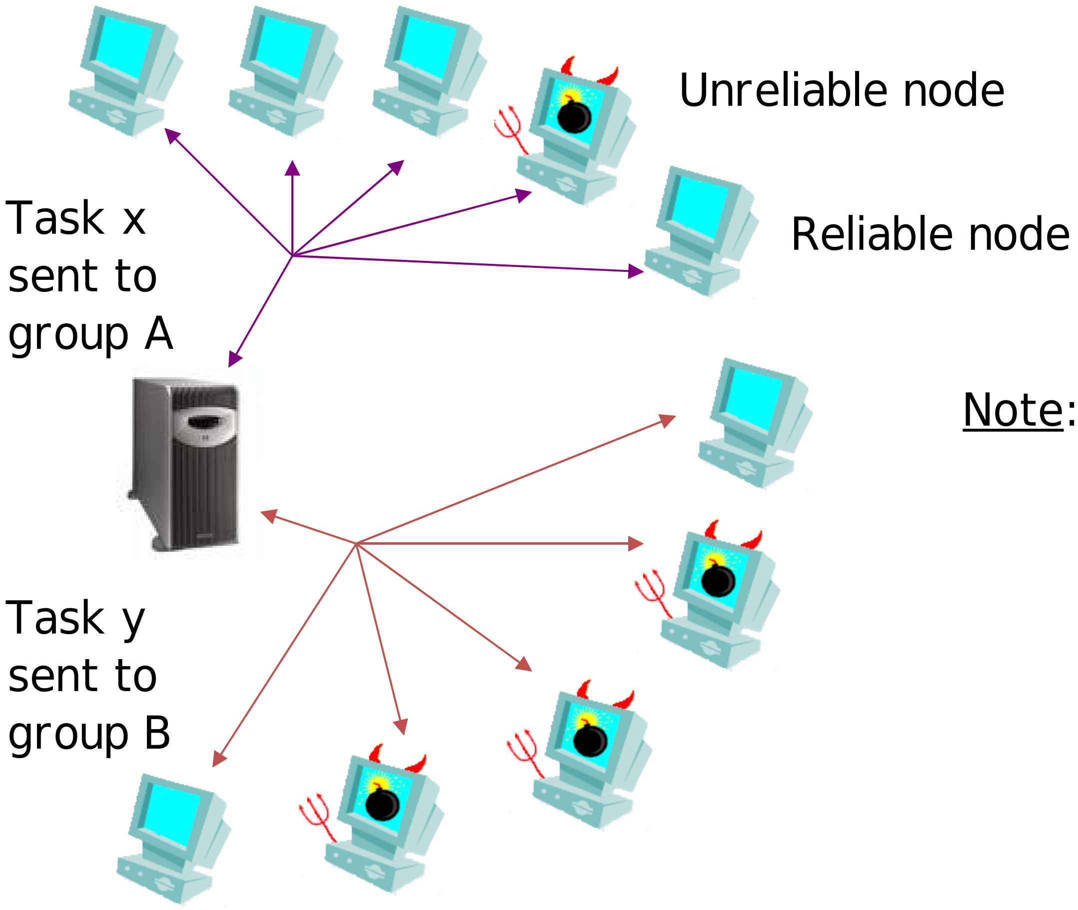
Replication Challenges

- How many replicas?
  - too many - waste of resources
  - too few - application suffers
- Most approaches assume ad-hoc replication
  - under-replicate: task re-execution (^ latency)
  - over-replicate: wasted resources (v throughput)
- Using information about the past behavior of a node, we can intelligently size the amount of redundancy

System Model



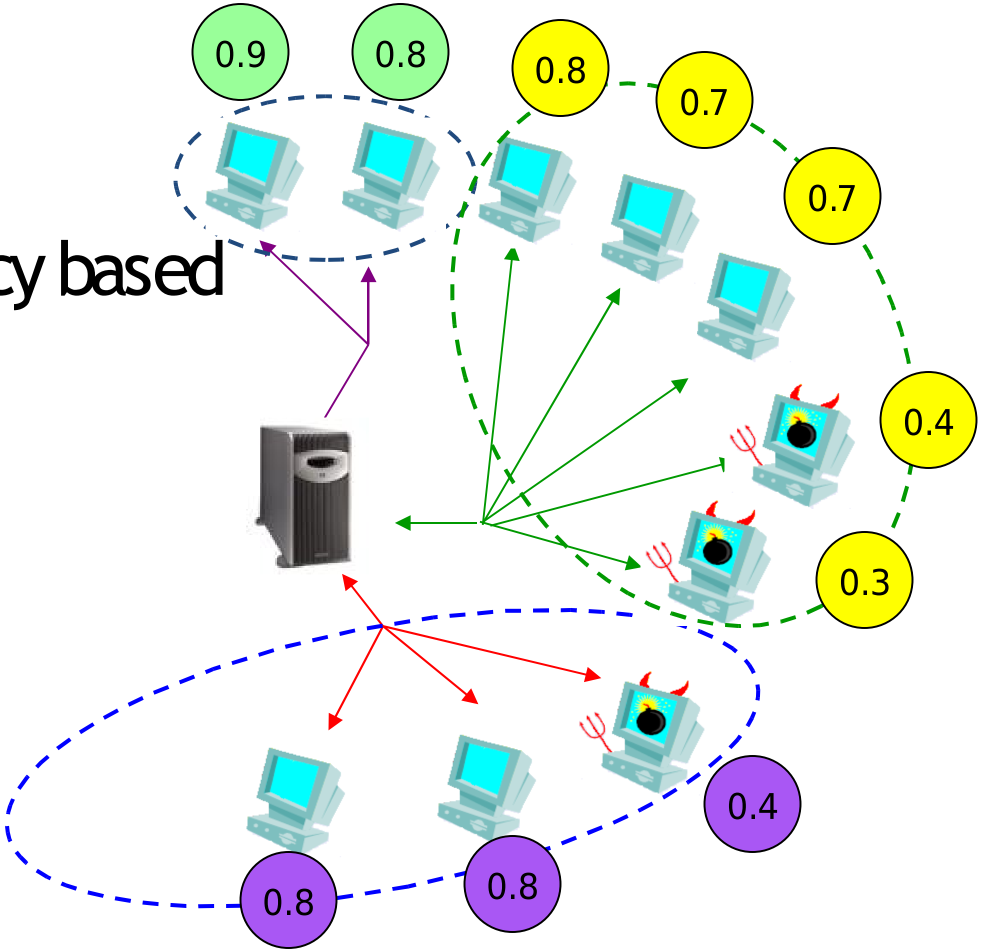
Problems with ad-hoc replication



Note: fixed size groups (size 5)

System Model

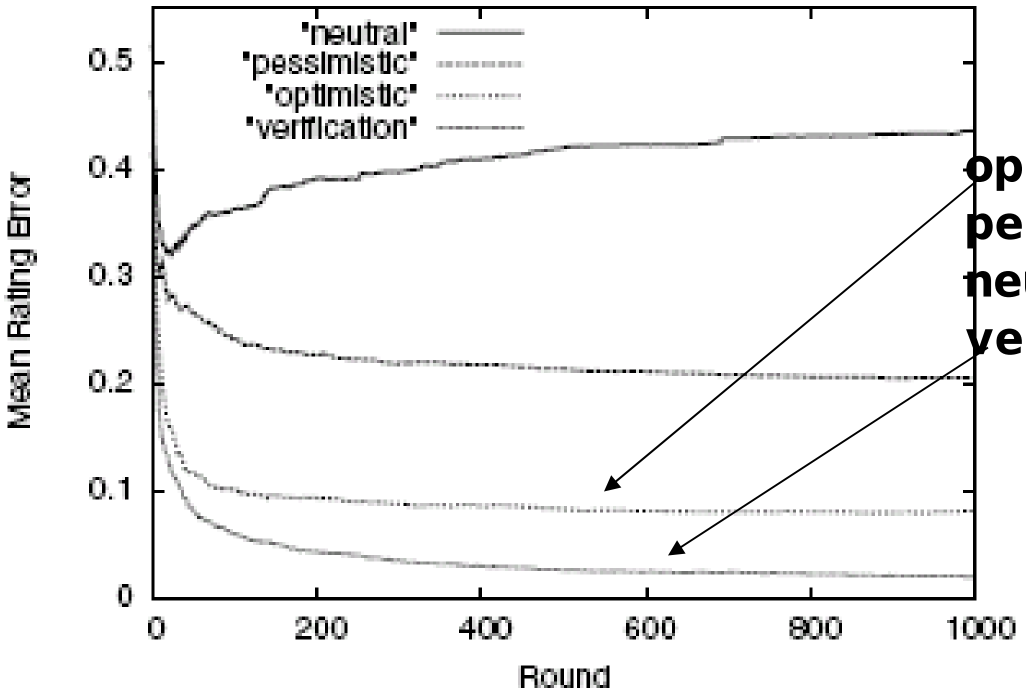
- Reputation rating r_i – degree of node reliability
- Dynamically size the redundancy based on r_i
- Note: variable sized groups



How to compute r_i?

$$r_i(t) = \frac{n_i(t) + 1}{N_i(t) + 2}$$

 # correct => but what is correct?
 # total tasks



optimistic: in the majority, ++, others --
pessimistic: no majority, -- to all
neutral: no majority, do nothing
verification: always right

Question

- What assumption is optimistic making?

Question

- What assumption is optimistic making?
  - majority is always right, hence no collusion

- This may be dangerous – can you think of examples?

Question

- What assumption is optimistic making?
  - majority is always right, hence no collusion
- This may be dangerous - can you think of examples?
  - Same virus attacks machines
  - A bad patch or code given to a set of machines

Smart Replication

- Reputation
  - ratings based on past interactions with clients
  - simple sample-based prob. (r_i) over window τ
  - extend to worker group (assuming no collusion) => likelihood of correctness (LOC)
- Smarter Redundancy
  - variable-sized worker groups
  - intuition: higher reliability clients => smaller groups

Terms

- LOC (Likelihood of Correctness), λ_g
  - computes the ‘actual’ probability of getting a correct answer from a group of clients (group g)

- Target LOC (λ_{target})
  - the success-rate that the system tries to ensure while forming client groups
  - related to the mean of the reliability distribution

LOC Functions

- Function for LOC

$$= \sum_{m=k+1}^{2k+1} \sum_{\left\{\varepsilon_1,\ldots\varepsilon_{2k+1}:\sum_{i=1}^{2k+1}\varepsilon_i=m\right\}} \prod_{i=1}^{2k+1} r_i^{\varepsilon_i}(1-r_i)^{1-\varepsilon_i}$$

- Function for lower bound of LOC

$$\geq \sum_{m=k+1}^{2k+1} \binom{2k+1}{m} \prod_{i=1}^{2k+1} r_i^{\alpha_m}(1-r_i)^{1-\alpha_m} \qquad \alpha_m = \frac{\binom{2k}{m-1}}{\binom{2k+1}{m}}$$

Performance Metrics

- Guiding metrics
  - throughput ρ: # of successfully completed tasks in an interval
  - success rate *s*: ratio of throughput to number of tasks attempted

Algorithm Space

- How many replicas?
  - algorithms compute how many replicas to meet a success threshold
  - first-, best-fit, random, fixed, ...

- How to reach consensus?
  - M-first (better for timeliness)
  - Majority (better for byzantine threats)

Scheduling Algorithms

- First-Fit
  - attempt to form the first group that satisfies the target LOC while considering the relative client ratings
- Best-Fit
  - attempt to form a group that best satisfies the target LOC while considering the relative client ratings
- Random-Fit
  - attempt to form a random group that satisfies the target LOC
- Fixed-size
  - randomly form fixed sized groups; ignore client ratings

Scheduling Algorithms

- Different size groups may result
- Order nodes by r_i

Algorithm 2 First-Fit (w worker-list, τ task-list, λ_{target} target LOC, R_{min} min-group-size, R_{max} max-group-size)

```
Sort the list w of all available workers on the basis of the reliability ratings r_i
while |w| ≥ R_min do
  Select task τ_i from τ
  repeat
    Assign the most reliable worker w_r from c to G_i
    w ← w - w_r
    Update λ_i
  until (λ_i ≥ λ_target AND |G_i| ≥ R_min) OR |G_i| = R_max
end while
```

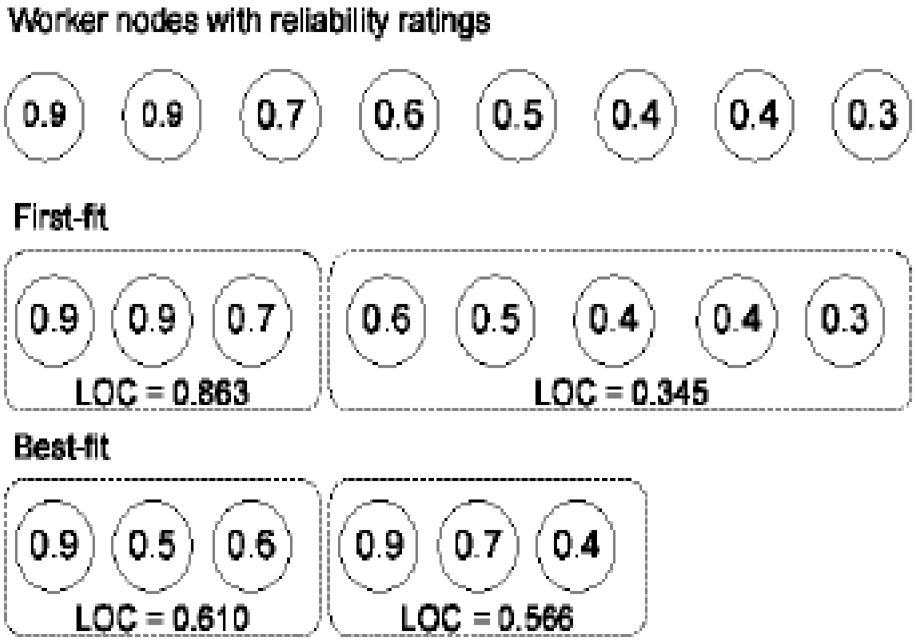
Scheduling Algorithms (cont'd)

Algorithm 3 Best-Fit (w worker-list, τ task-list, λ_{target} target LOC, R_{min} min-group-size)

Sort the list w of all available workers on the basis of the reliability ratings r_i
while $|w| \geq R_{min}$ **do**
  Select task τ_i from τ
  Search for a set s of n workers w_n from w such that λ_s exceeds λ_{target} minimally
  if such a set s is found **then**
    Assign the w_n workers to G_i
  else
    Select the set of n workers s for which $\lambda_{target} - \lambda_s$ is minimized
    Assign the w_n workers to G_i
  end if
  $w \leftarrow w - w_n$
end while

Different Groupings

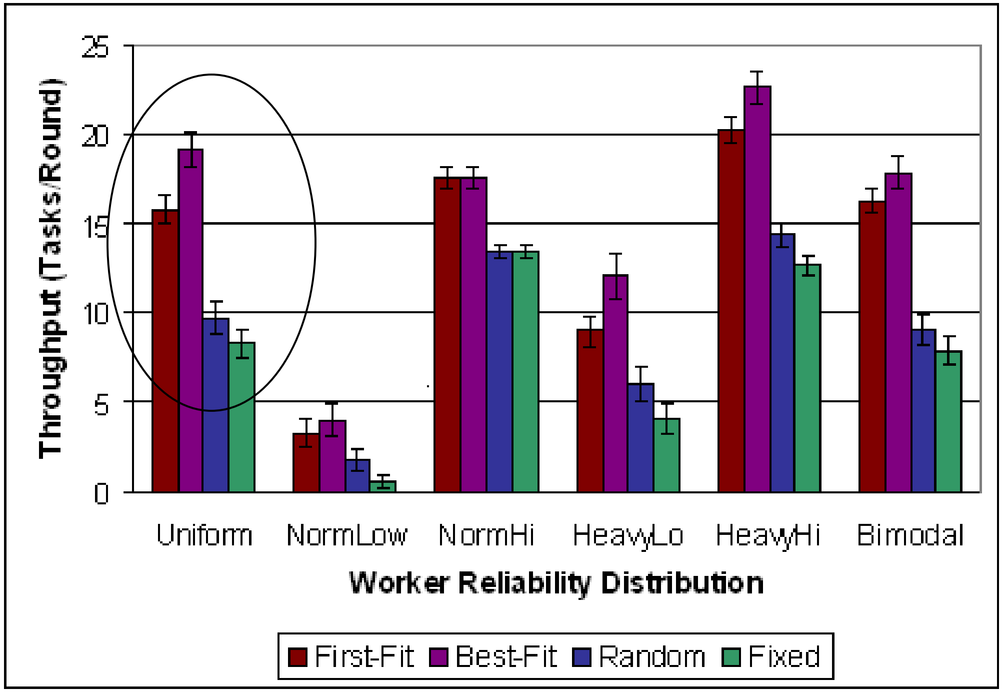
$\lambda_{target} = .5$

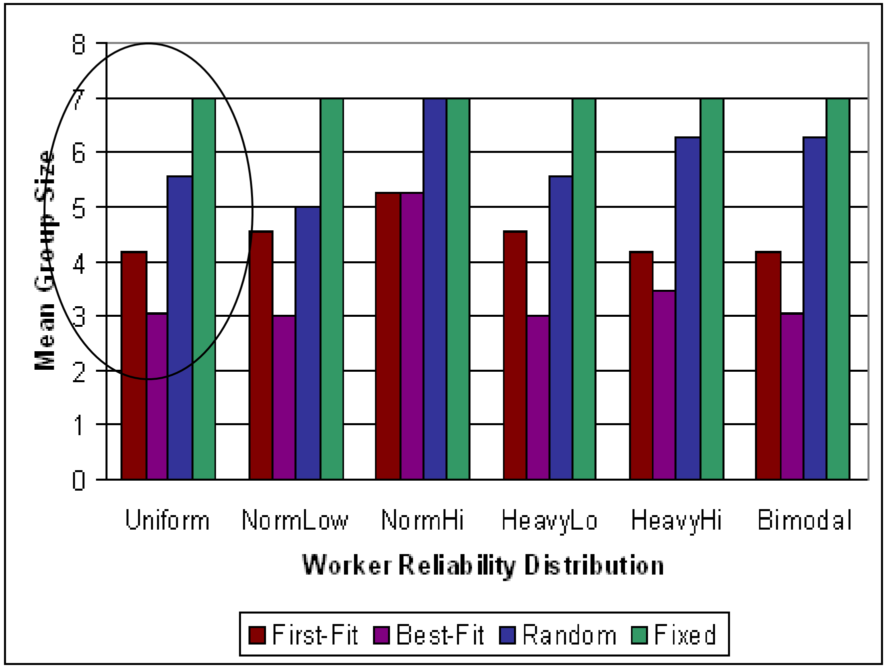


Evaluation

- Baselines
  - **Fixed** algorithm: statically sized equal groups uses no reliability information
  - Random algorithm: forms groups by randomly assigning nodes until λ_{target} is reached
- Simulated a wide-variety of node reliability distributions

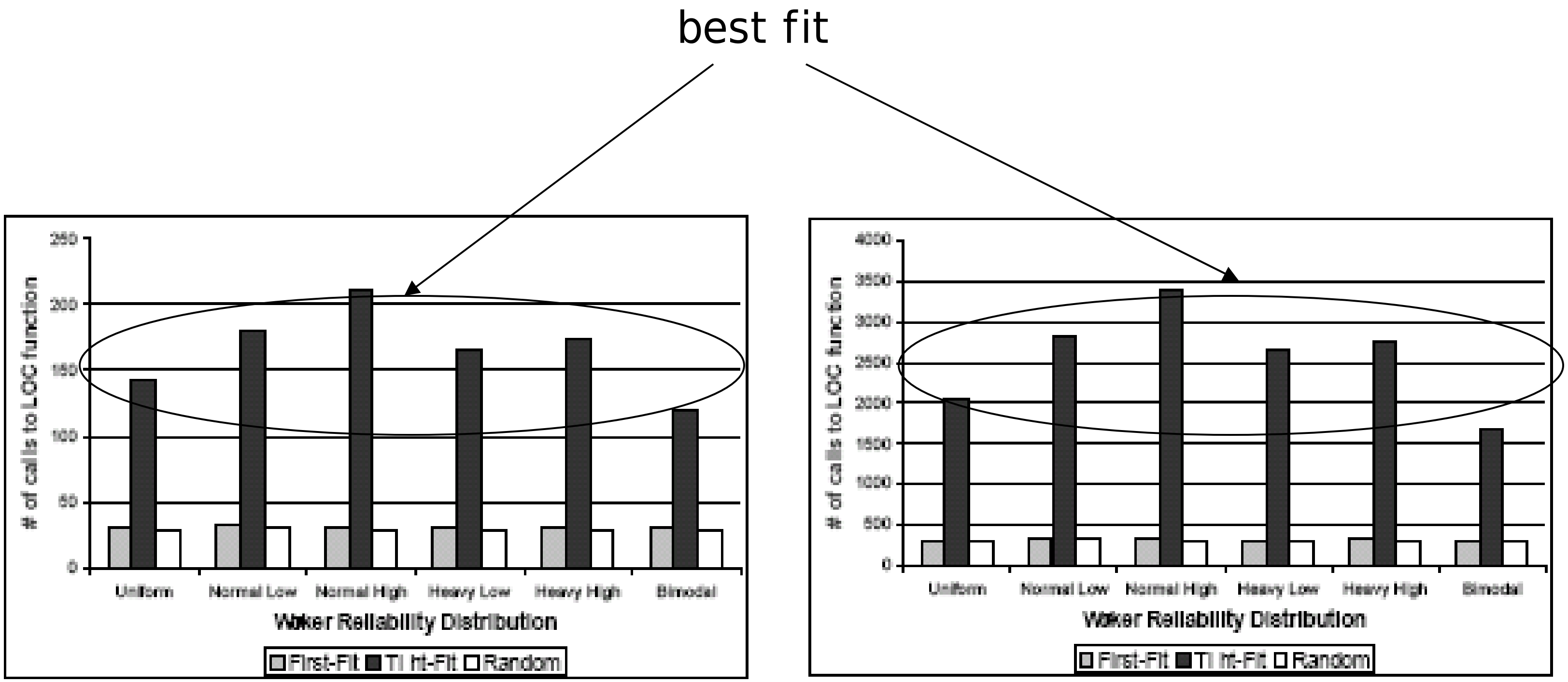
Experimental Results: correctness





This was a simulation based on byzantine behavior … majority voting

Overhead

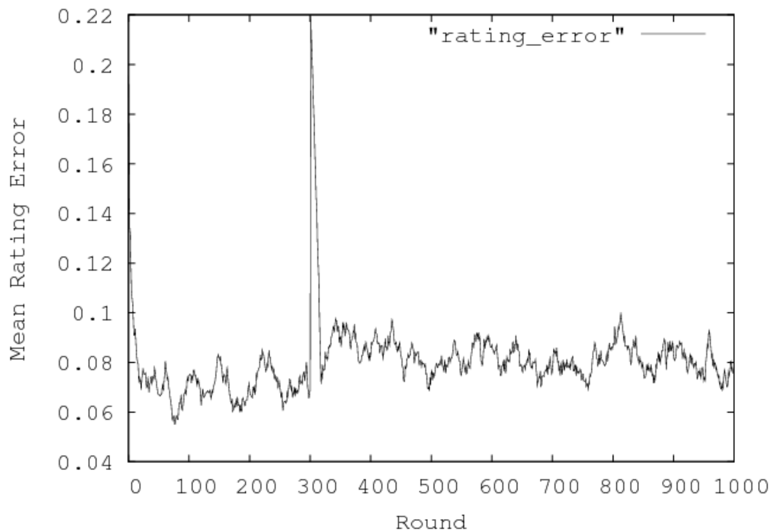


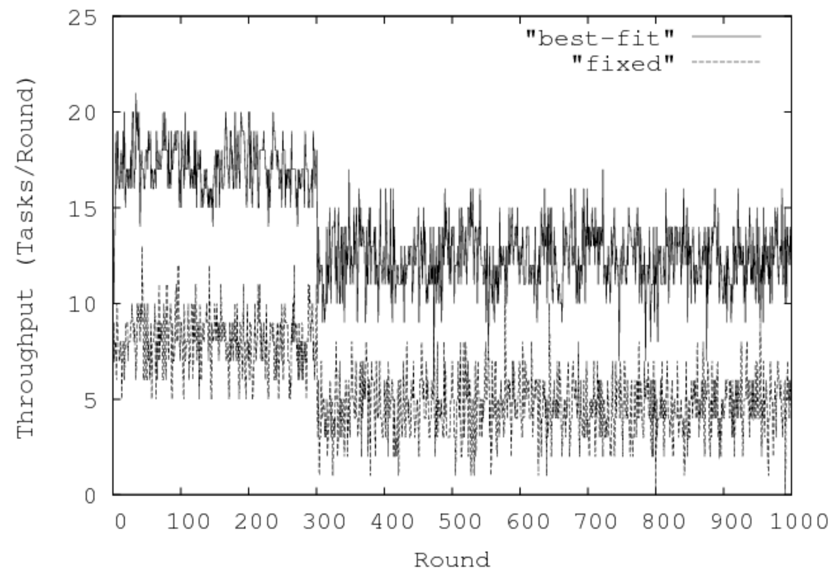
(a)Network size=100

(b)Network size=1000

Non-stationarity

- Nodes may suddenly shift gears
  - deliberately malicious, virus, detach/rejoin
  - underlying reliability distribution changes
- Solution
  - window-based rating (reduce $\tau\Theta$ from infinite)
  - adapt/learn λ_{target}
- Experiment: “blackout” at round 300 (30% effected)





Role of λ_{target}

- Key parameter
- Too large
  - groups will be too large (low throughput)
- Too small
  - groups will be too small (low success rate)
- Adaptively learn it by maximizing $\rho * s$
  - "goodput"
  - or could bias toward ρ or s

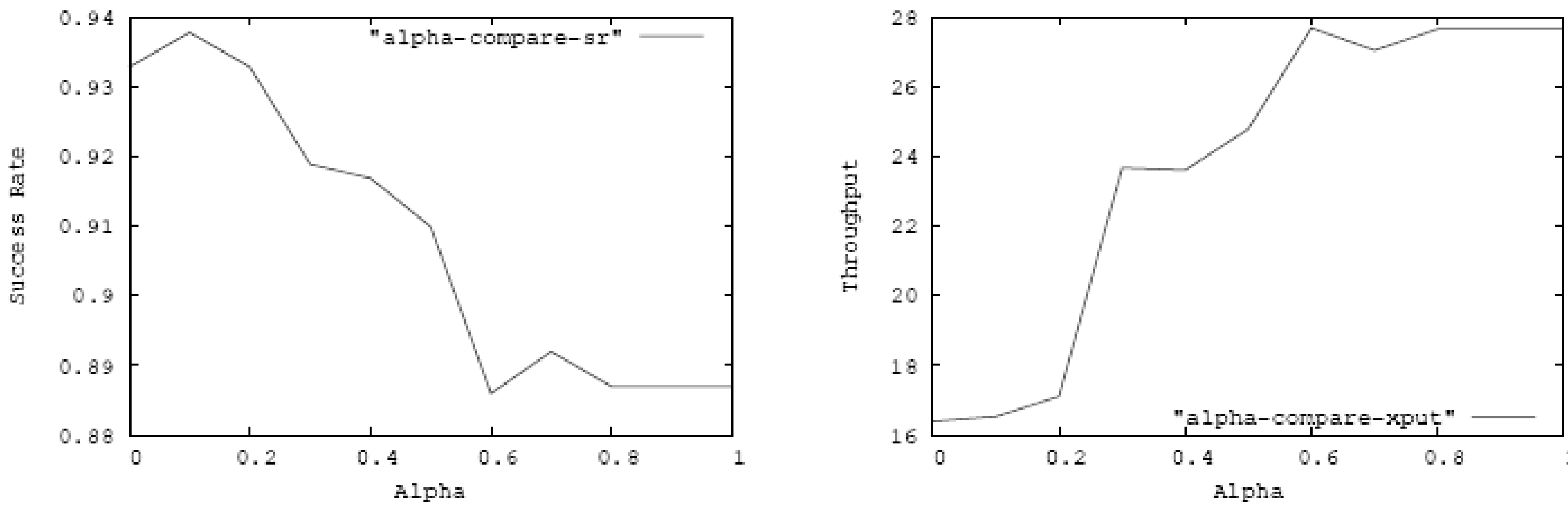
Adaptive algorithm

- Multi-objective optimization
  - choose target LOC to simultaneously maximize throughput ρ and success rate *s*
  - use weighted combination to reduce multiple objectives to a single objective
  - employ hill-climbing and feedback techniques to control dynamic parameter adjustment

Adaptive Algorithm

throughput success rate

$$G(\rho, s) = \alpha \cdot \rho + (1 - \alpha) \cdot s,$$



(a) Success Rate (b) Throughput

Fig. 9. Comparison of throughput/success rate achieved using adaptive algorithm with varying α

Algorithm 1 UpdateTargetLOC (λ_{target} target LOC, α throughput-weight)

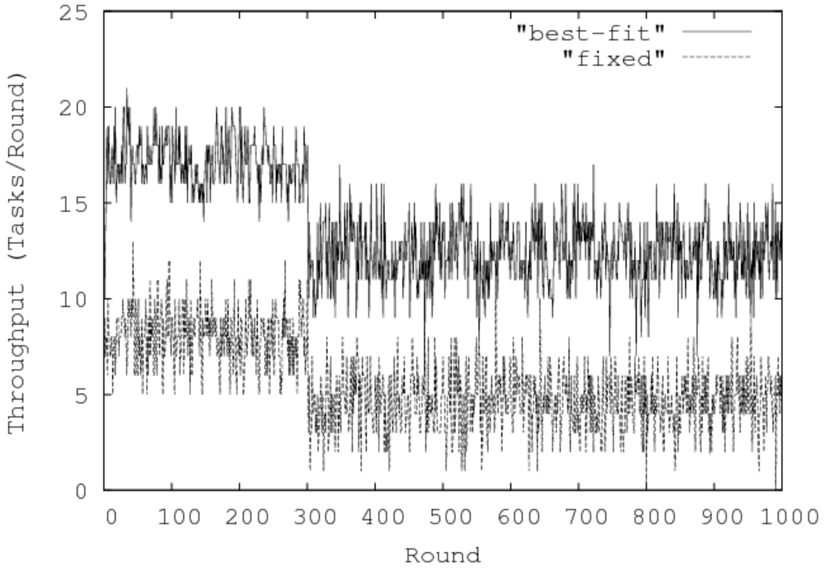
```
Local variables: s state, d direction

if (round % p) = 1 ... p-1 then
   Update measures of mean normalized throughput xp and success rate sr
else
   if s = CONVERGING then
      if round = p then
         Set initial direction d based on mean client reliability
         j ← 4
      end if
      G_last ← G
      Gain G ← α * xp + (1-α) * sr
      if G / G_last ≥ delta_mod then
         j ← j−1
         Switch direction d
         if j = 0 then
            s ← STEADY-STATE
         end if
      else if G > G_avg OR G / G_last ≤ delta_in then
         if d = left then
            λ_target ← λ_target - 0.01*j
         else
            λ_target ← λ_target + 0.01*j
         end if
      else
         λ_target unchanged
         if λ_target unchanged for maxrounds rounds then
            s ← STEADY-STATE
         end if
      end if
   else
      Gain G ← α * xp + (1-α) * sr
      if G / G_last ≥ delta_sig then
         s ← CONVERGING, j ← 4
      end if
      G_avg ← weight_curr * G + weight_hist * G_avg
   end if
end if
```

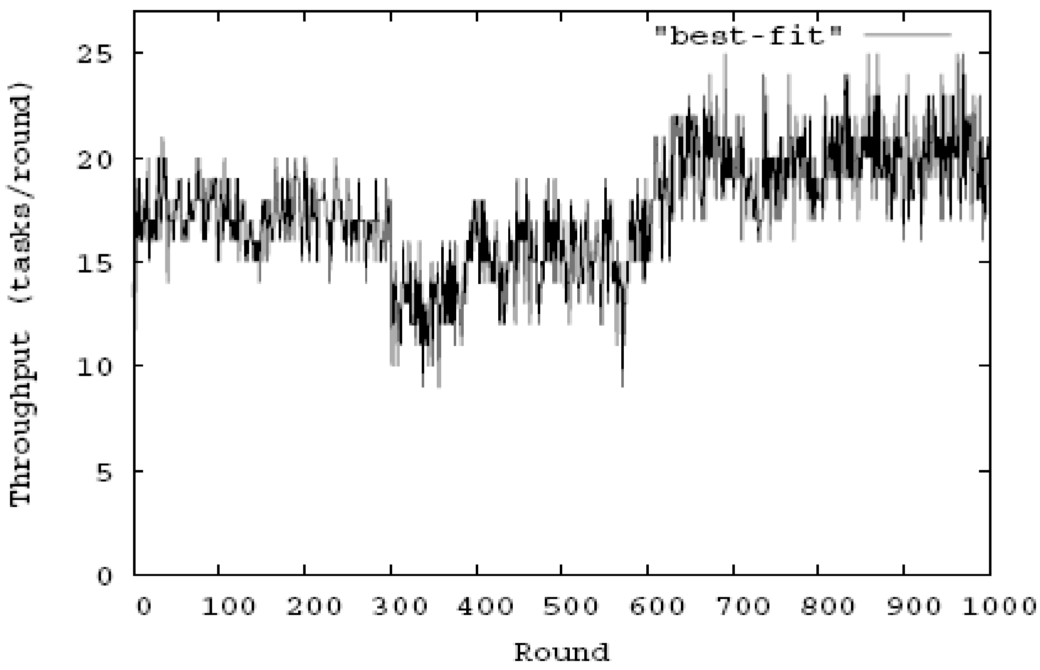
throughput   success rate

$$G(\rho, s) = \alpha \cdot \rho + (1 - \alpha) \cdot s,$$

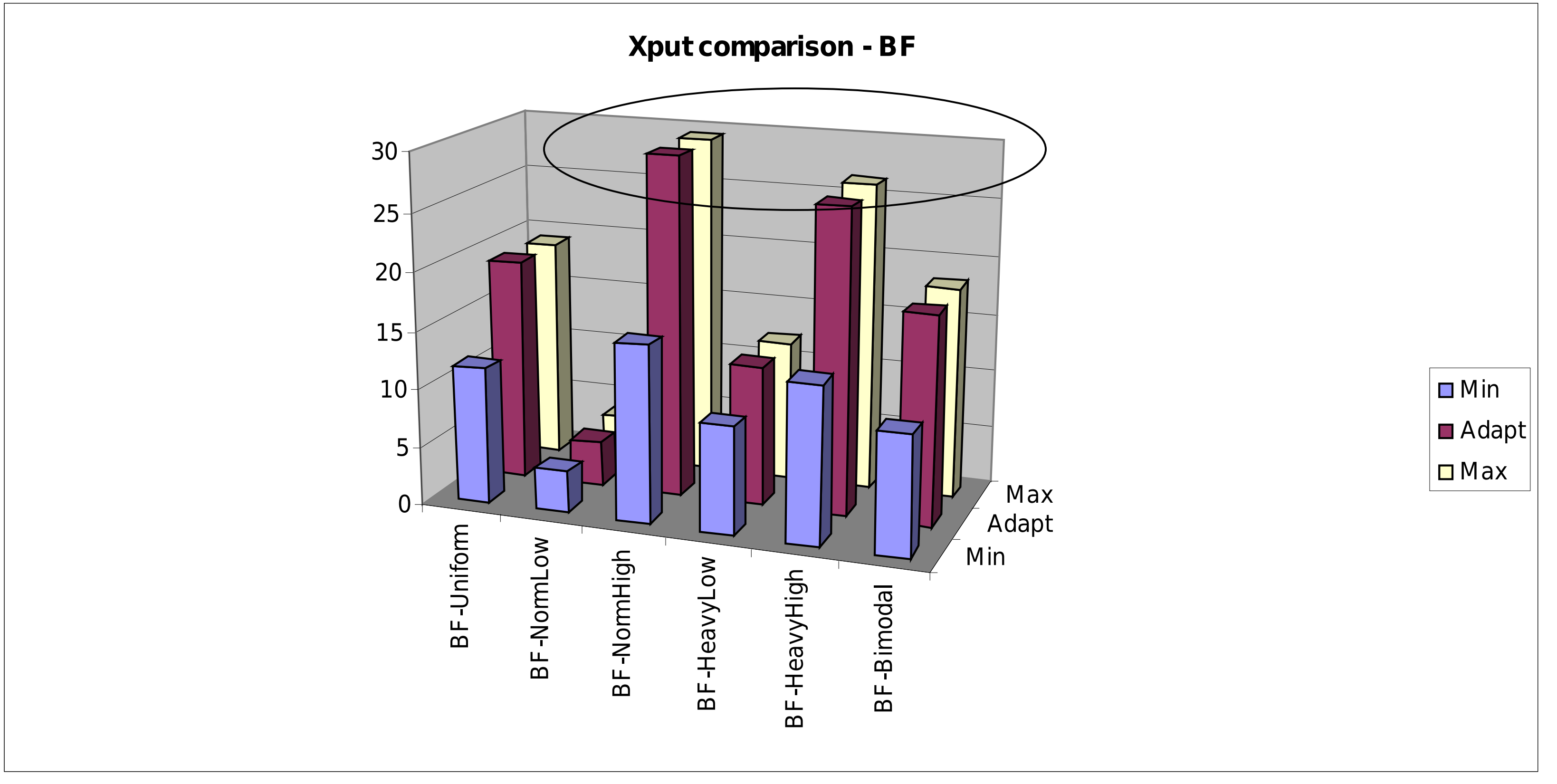
Adapting λ_{target}

- Blackout example

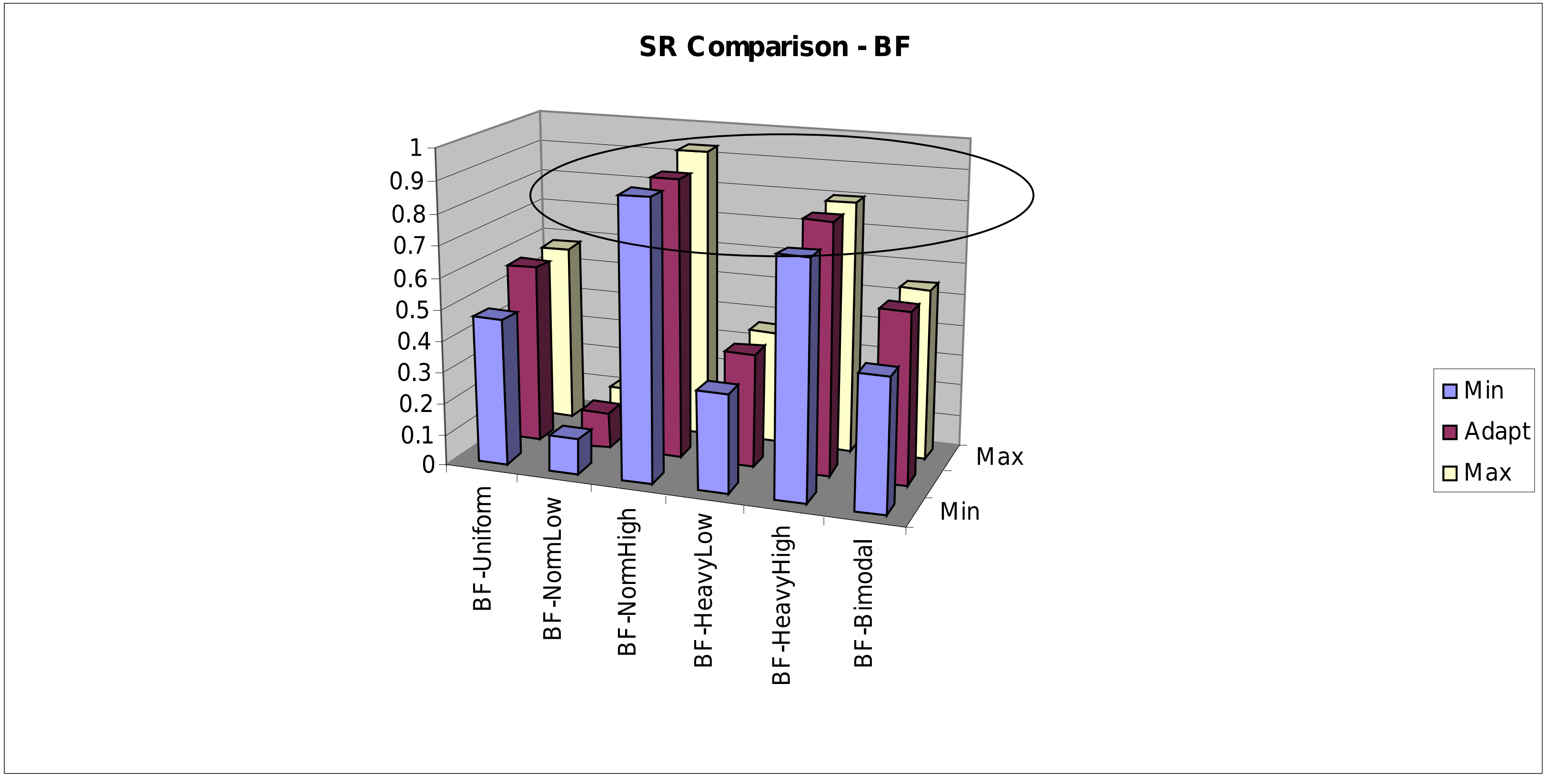




Going Parameterless: Throughput



Success-rate

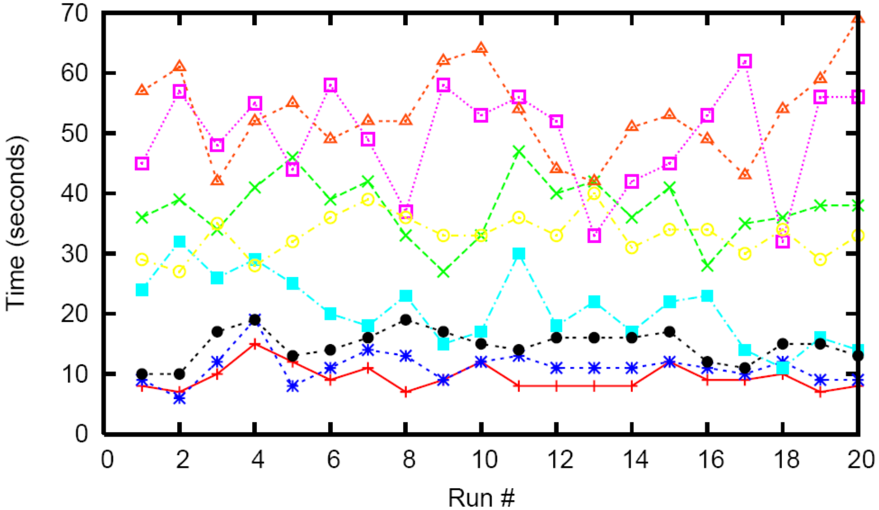


convergence rate: 150-350 rounds

Timeliness

- What is timeliness?
- Two viewpoints
  - (1) soft deadlines
    - user interaction, visualizing output from computation
  - (2) hard deadlines
    - need to get X results done before HPDC/NSDI/... deadline
- Start with (1)
- Live experimentation on PlanetLab

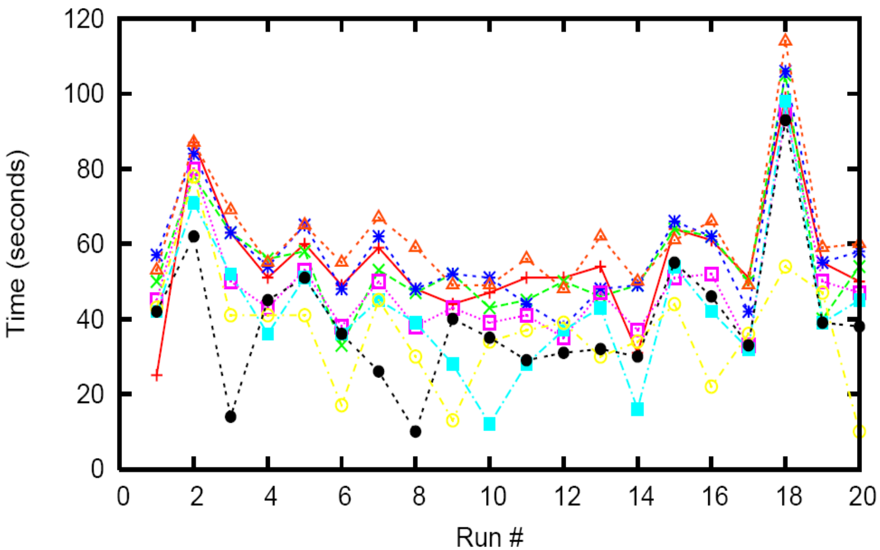
Some PL data



Computation Heterogeneity

- both across and within nodes

PlanetLab – lower bound



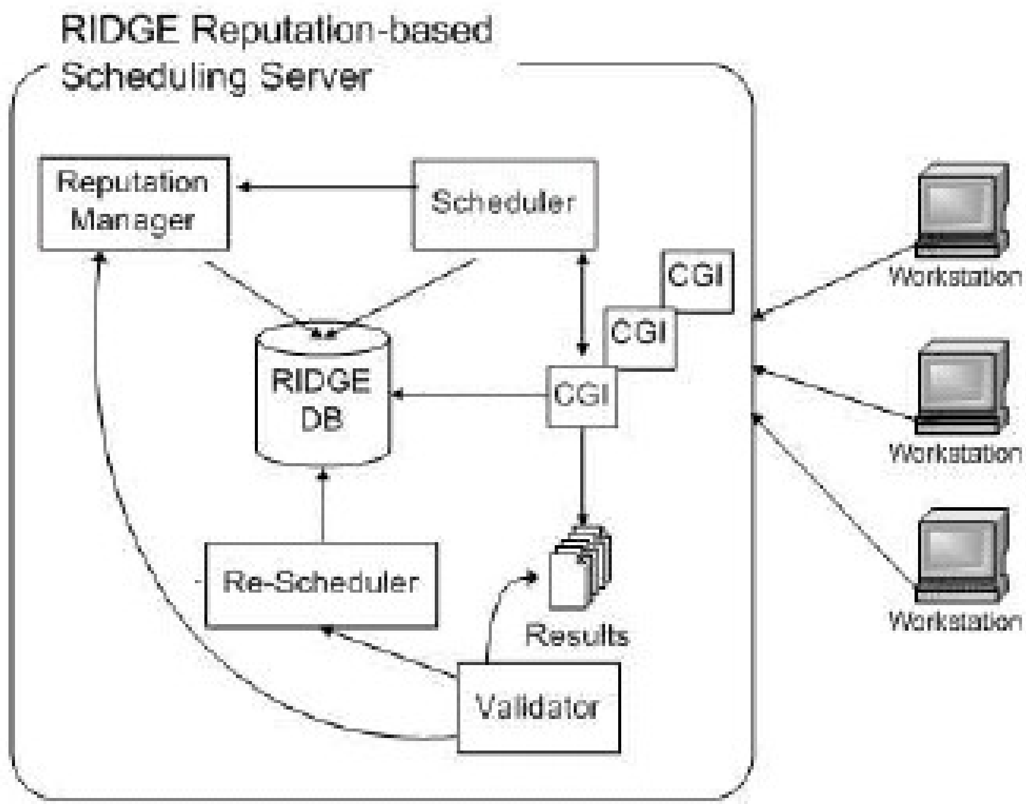
Communication Heterogeneity

- both across and within nodes

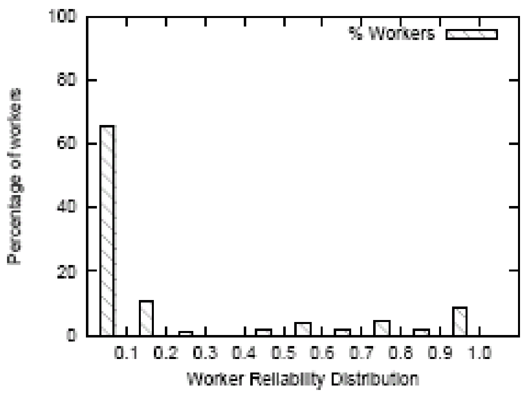
RIDGE

- RIDGE scheduling framework on top of BOINC
- PL Grid – **120** dispersed worker nodes
- Test application – BLAST, a Bioinformatics application for genomic sequence comparison
- M-First (M=1) consensus
- Assume fully correct

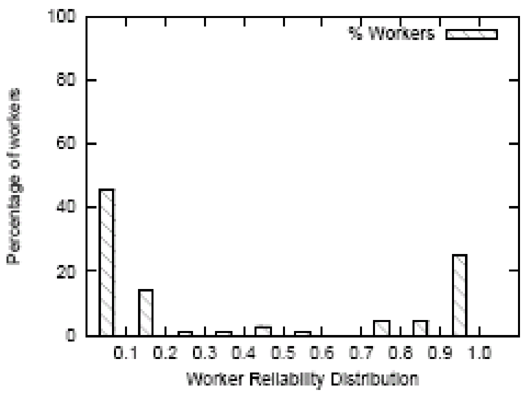
RIDGE Scheduler



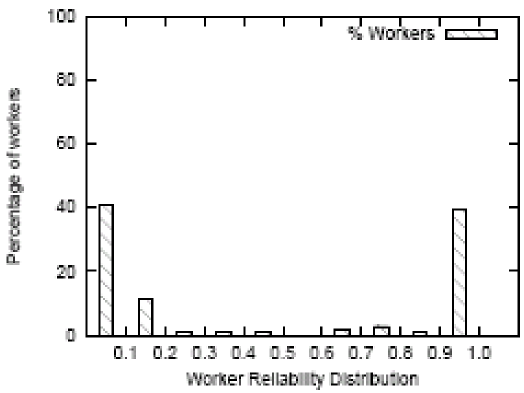
PL Timeliness Trace + Soft deadlines



(a) Low Reliability Environment (LowRE): Execution-Threshold=120s



(b) Moderate Reliability Environment (ModRE): Execution-Threshold=180s



(c) High Reliability Environment (HighRE): Execution-Threshold=240s

reliability = # tasks completed by deadline/total tasks

Learning Rate

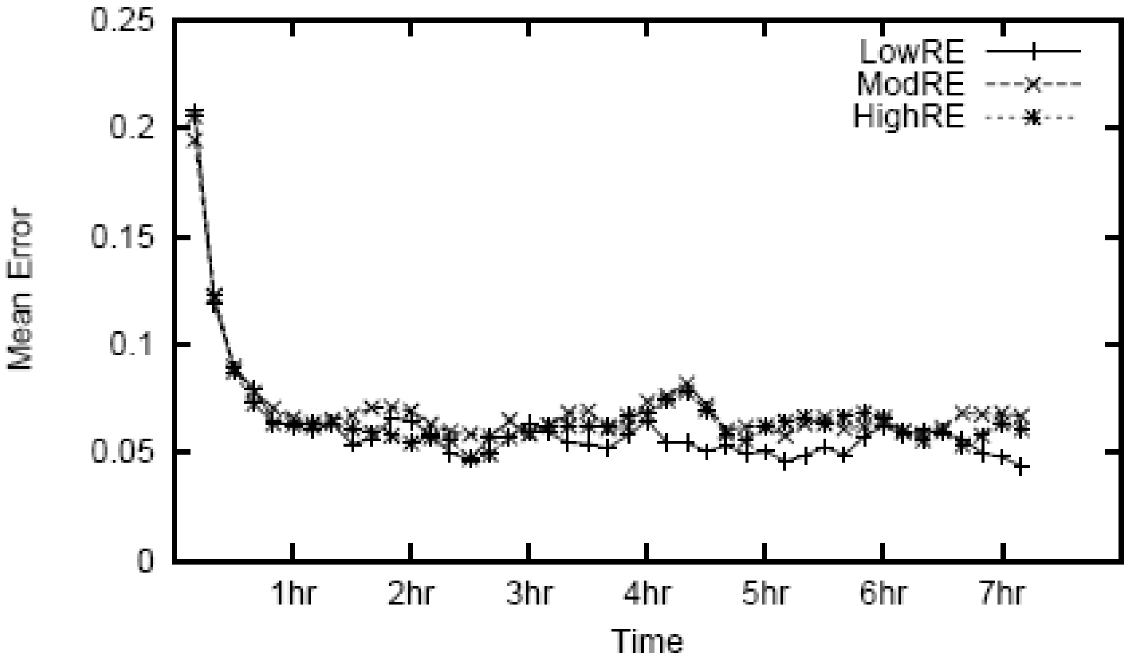


Figure 7: Learning Behavior of RIDGE

Performance of BOINC: Timeliness

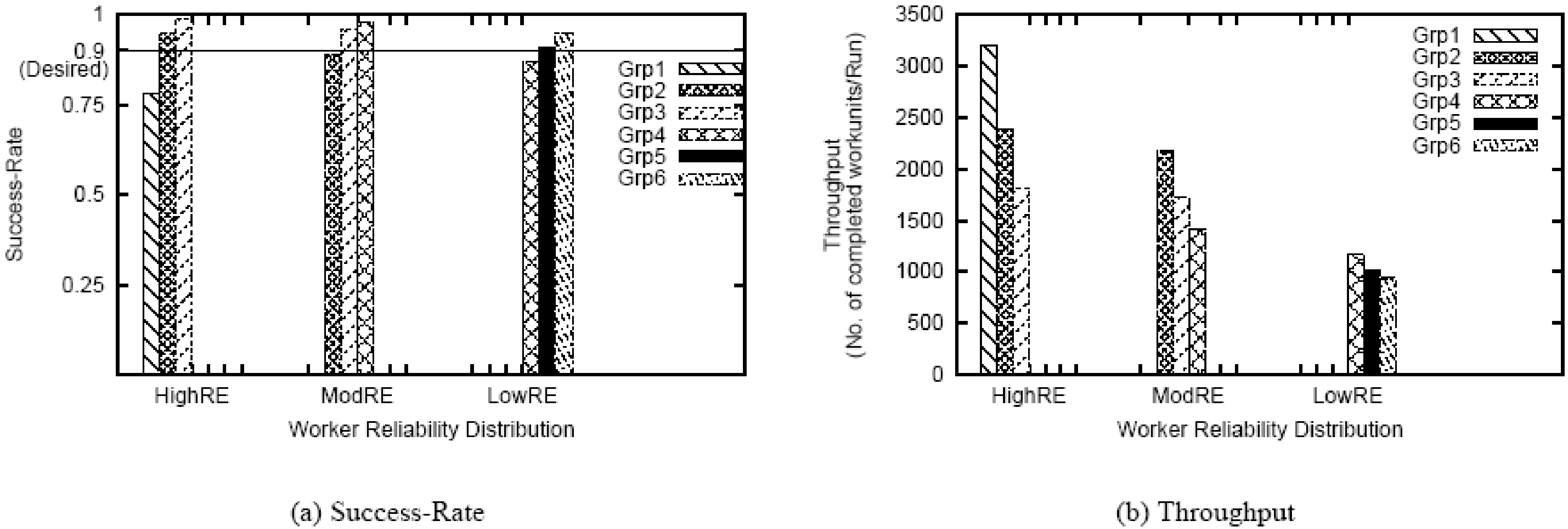
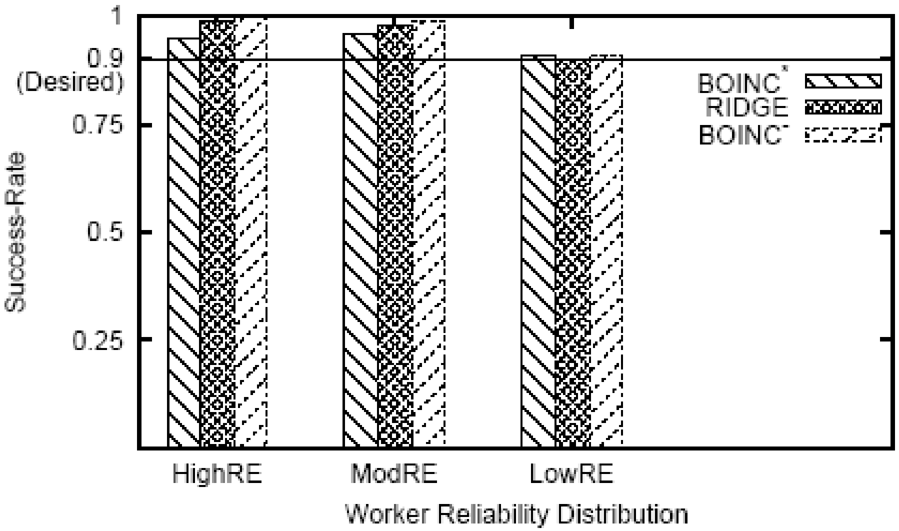


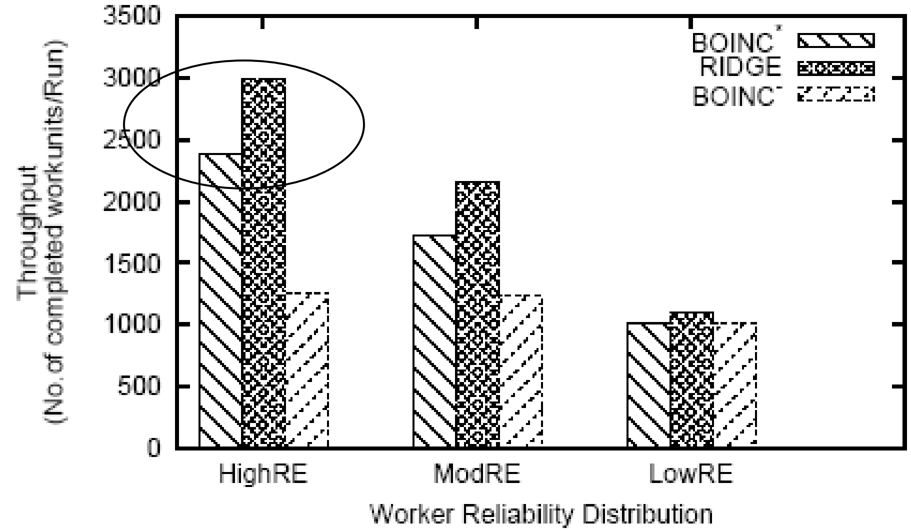
Figure 8: Comparison of different BOINC confi gurations.

best replication factor varies

Experimental Results: timeliness



(a) Success-Rate



(b) Throughput

M-first (M=1), best BOINC (BOINC*), conservative (BOINC-) vs. RIDGE

Makespan: RIDGE vs. BOINC (task level)

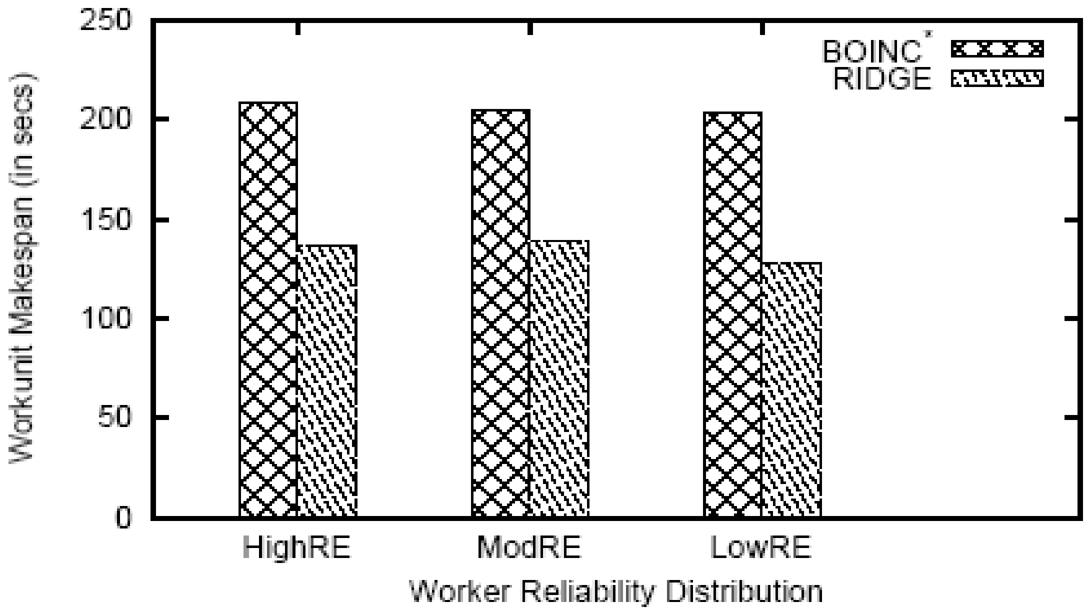
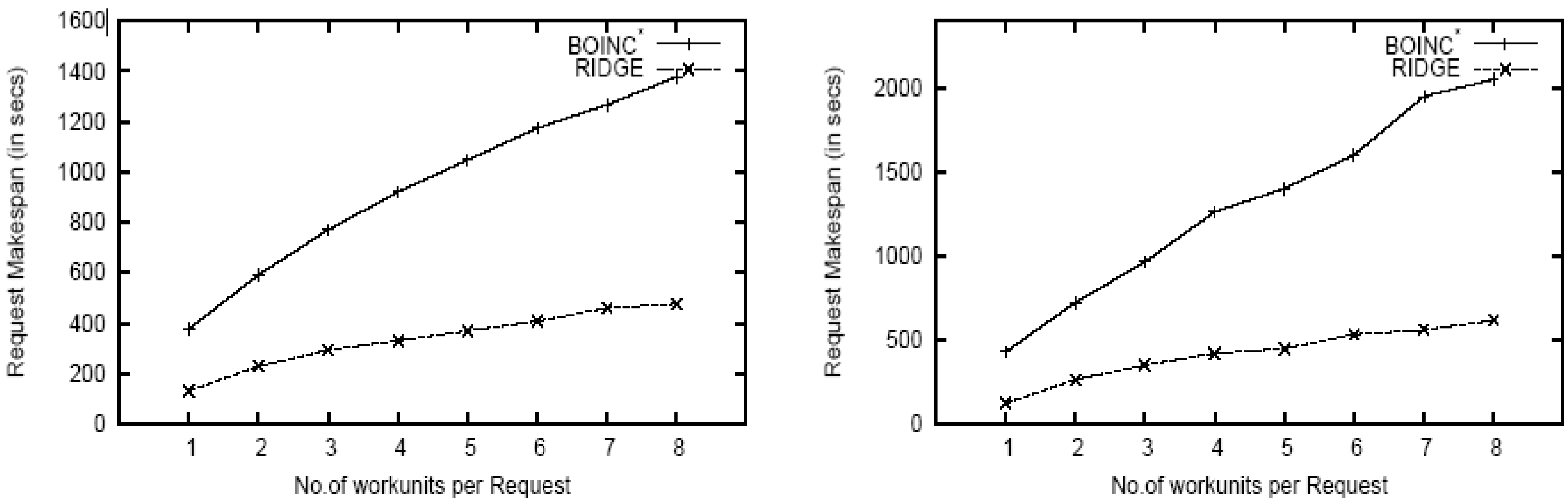


Figure 5: Makespan Comparison

Makespan Comparison (application level)



(a) HighRE Makespan

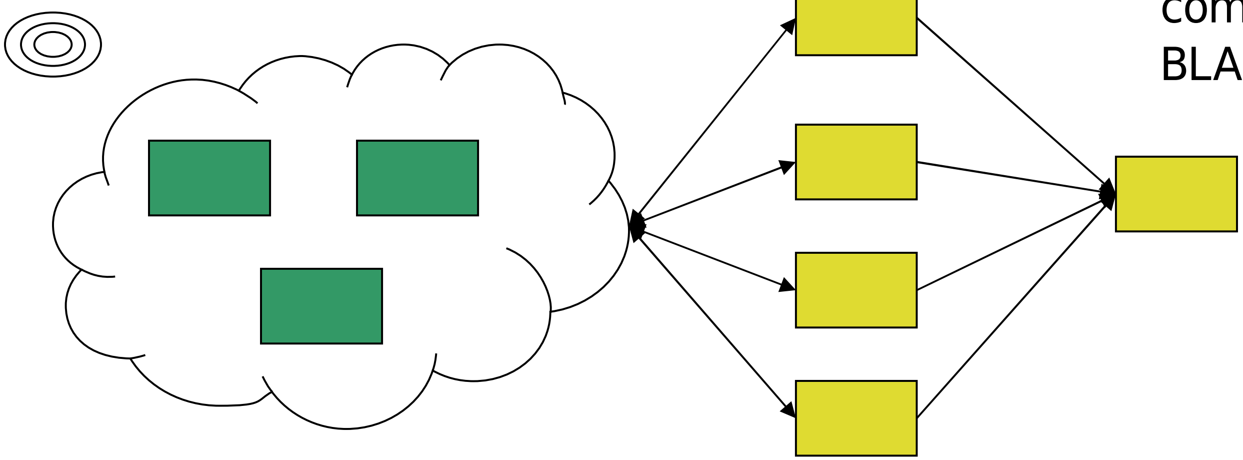
(b) LowRE Makespan

Figure 11: Comparison of Request Makespan for different reliability environments

Future Work

- Mechanisms to retain node identities (hence r_i) under node churn
  - "node signatures" that capture the characteristics of the node
- Combine timeliness + correctness
- Client collusion
  - Detection: group signatures
  - Prevention:
    - Uses quizzes (ground-truth); server should run tasks occasionally
    - random group formation

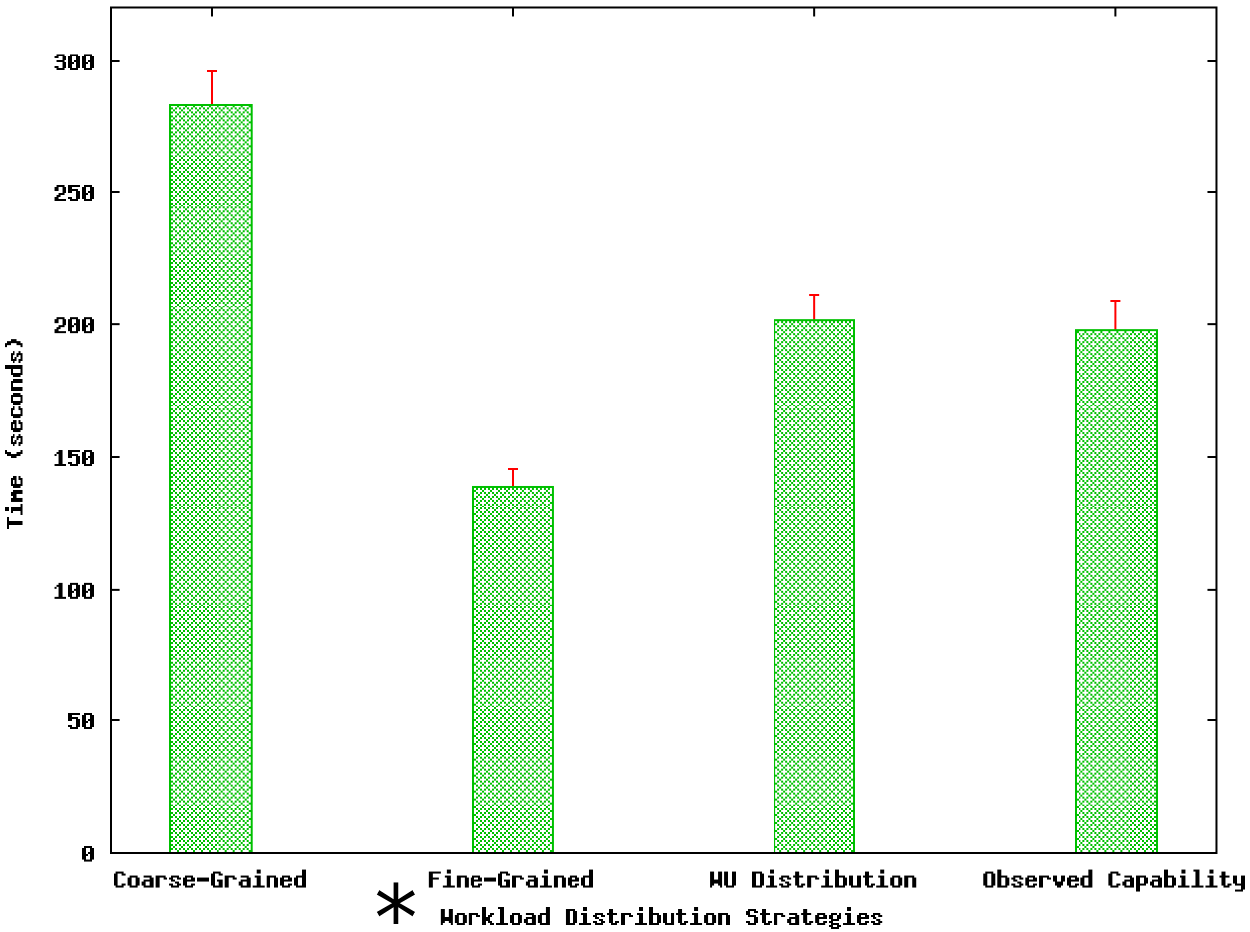
Computational Makespan



compute dominates:
BLAST

Reducing makespan requires not only smart redundancy and scheduling, but smart **decomposition**

Reducing Makespan



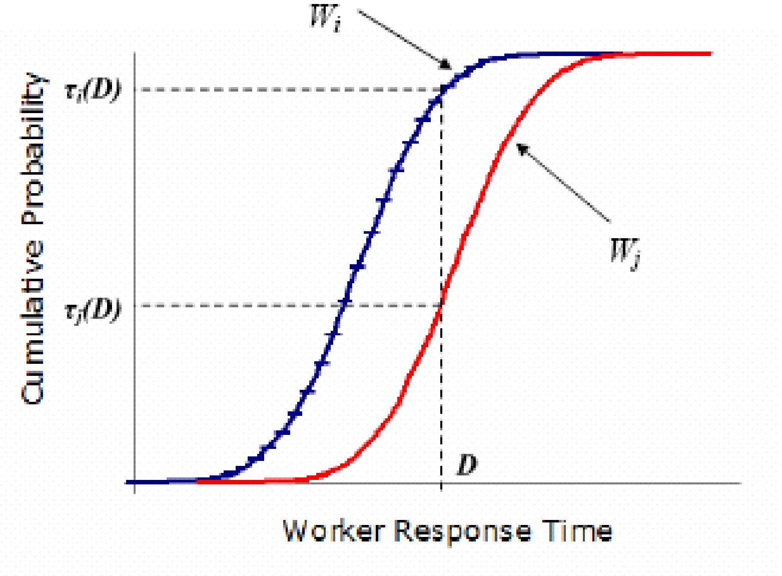
*

equal-sized variable-sized

Hard Deadlines

- Sometimes deadlines are hard
- The task which simulates a key parameter that finished after HPDC deadline may not be useful
- Many other real-time scenarios
- Challenge is again the time variability of open distributed systems

Timeliness Definition



Timeliness: $\tau_i(D) = CDF_i(D)$

Group timeliness: $\tau_G(D) = 1 - \prod_{i=1}^{n}(1 - \tau_i(D))$

Redundancy

- Two workers W1 and W2 with timeliness of .8 and .6 respectively for D=100
  - Objective is .9 reliability for a task (TSR)
  - Can't do it alone
- Assign both to the task => .92

$$\tau_G(D) = 1 - \prod_{i=1}^{n}(1 - \tau_i(D))$$

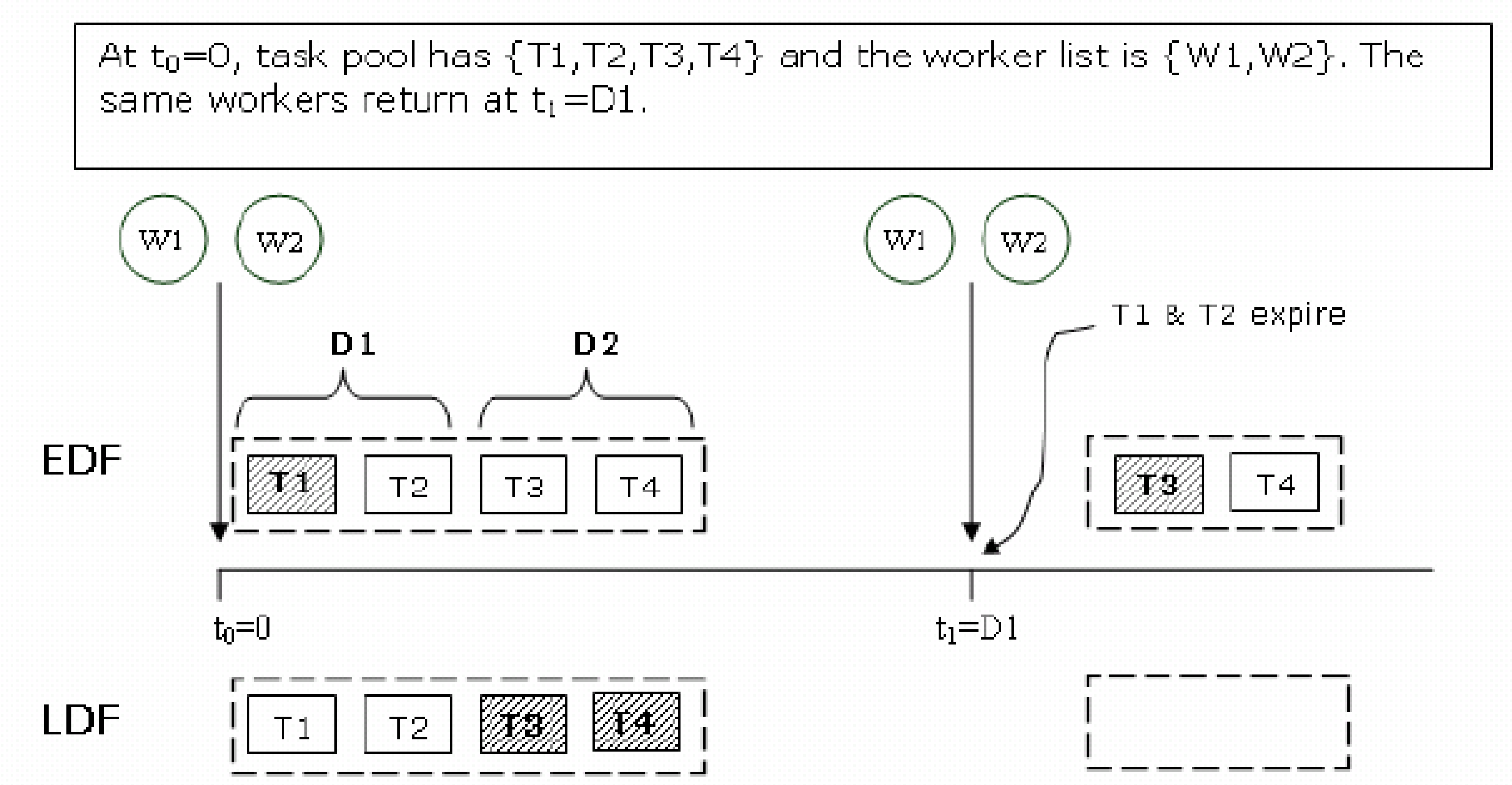
EDF (earliest) vs. LDF (latest)

- Tasks T1, T2, T3 with successively higher deadlines
- Three workers W1, W2, W3
- T1 has D1 and needs all three workers
- T2 and T3 have D2 (D2 > D1), any single worker will do)

- EDF: pick T1; T2 and T3 cannot run, tput = 1
- LDF: picks T2 and T3, tput = 2
- Coupling redundancy and deadlines is tricky

EDF vs. LDF

- LDF has good tput, but is it fair?
- T1, T2 have D1 and T3, T4 have D2 (= 2 * D1)
- Need both workers to meet D1 but either for D2



- Both have a tput of 2 but LDF is unfair to short D

The Metrics

fairness: $$FI = \frac{\left[\sum_{i=1}^{m} x_i\right]^2}{m \sum_{i=1}^{m} {x_i}^2} \quad \text{where} \quad x_i = \frac{X_i}{C_i}$$

~ of tasks completed in bin i
# of tasks in bin i

tput = ~ of tasks completed by their deadline

Generalization

- LRED
  - Limited Resource Scheduling Earliest Deadline
- Balance
  - Tasks requiring more nodes (shorter deadlines) reduce tput
  - Favoring tasks with shorter deadlines improves fairness (doesn't starve longer deadline tasks since their deadlines get smaller with time!)

Intuitive Behavior

- Pick task groups with smaller resource requirements (to a point), larger D => tput

- Within each group of tasks, sort by earliest deadline => fairness

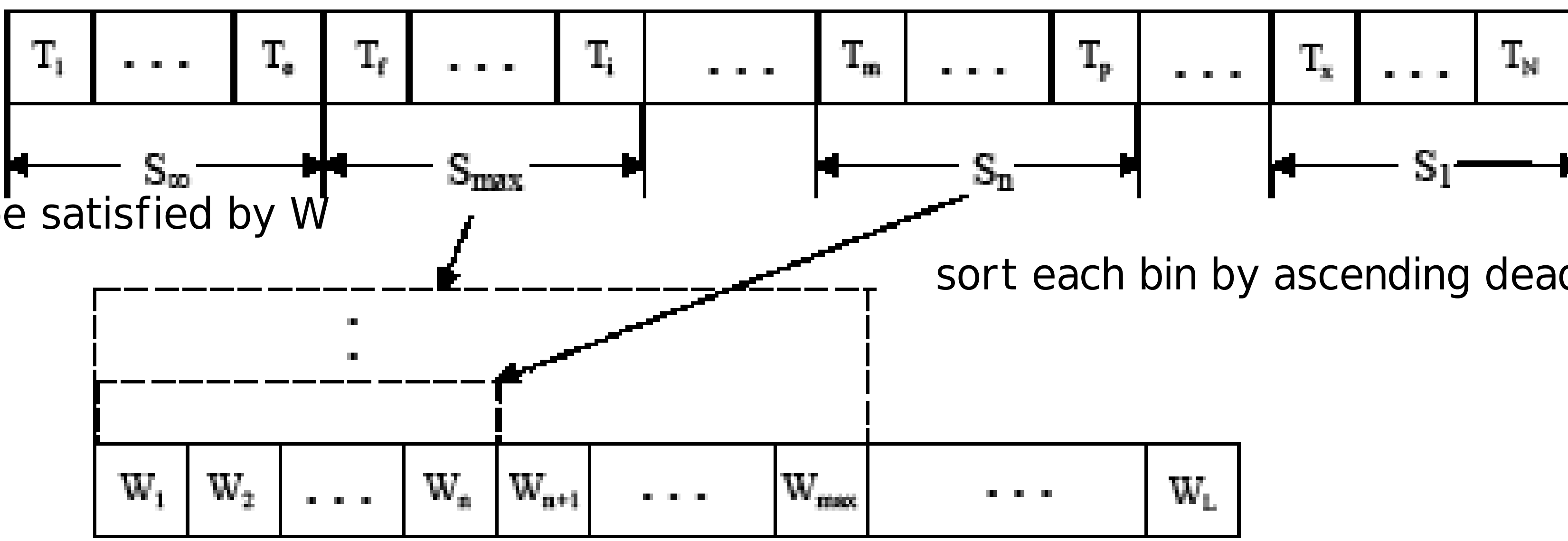
The Idea

N : Total no. of tasks in the task pool
L : Total no. of workers available at the scheduler

S_k : Set of tasks with ascending deadline values each of which can be completed with probability TSR by the workers $\{W_1,\ldots..W_k\}$ for k = 1,2,….,max. S_k could be empty.

S_∞ = Set of tasks that cannot be completed with probability TSR by any number of workers in $\{W_1,\ldots..W_L\}$

max = # of workers required by any task

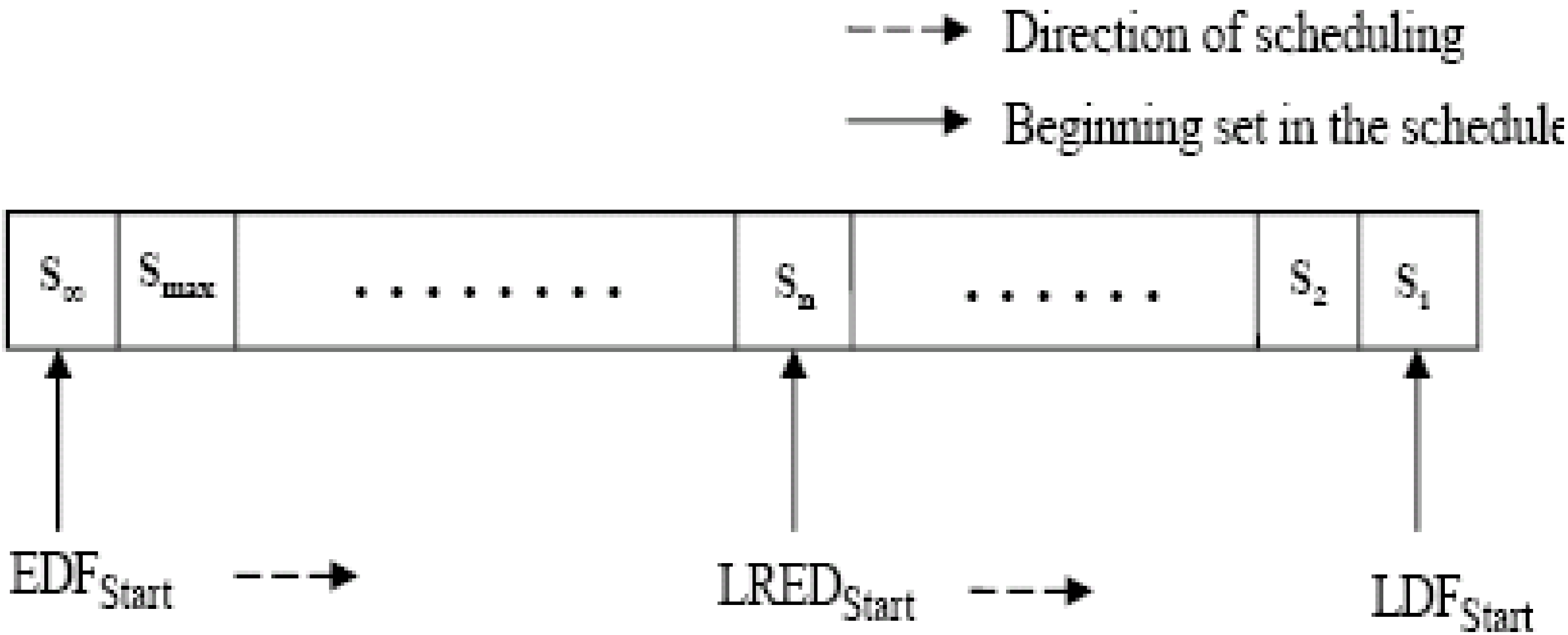


can't be satisfied by W

sort each bin by ascending deadline

sort by timeliness – use the mean

Algorithm Start Points



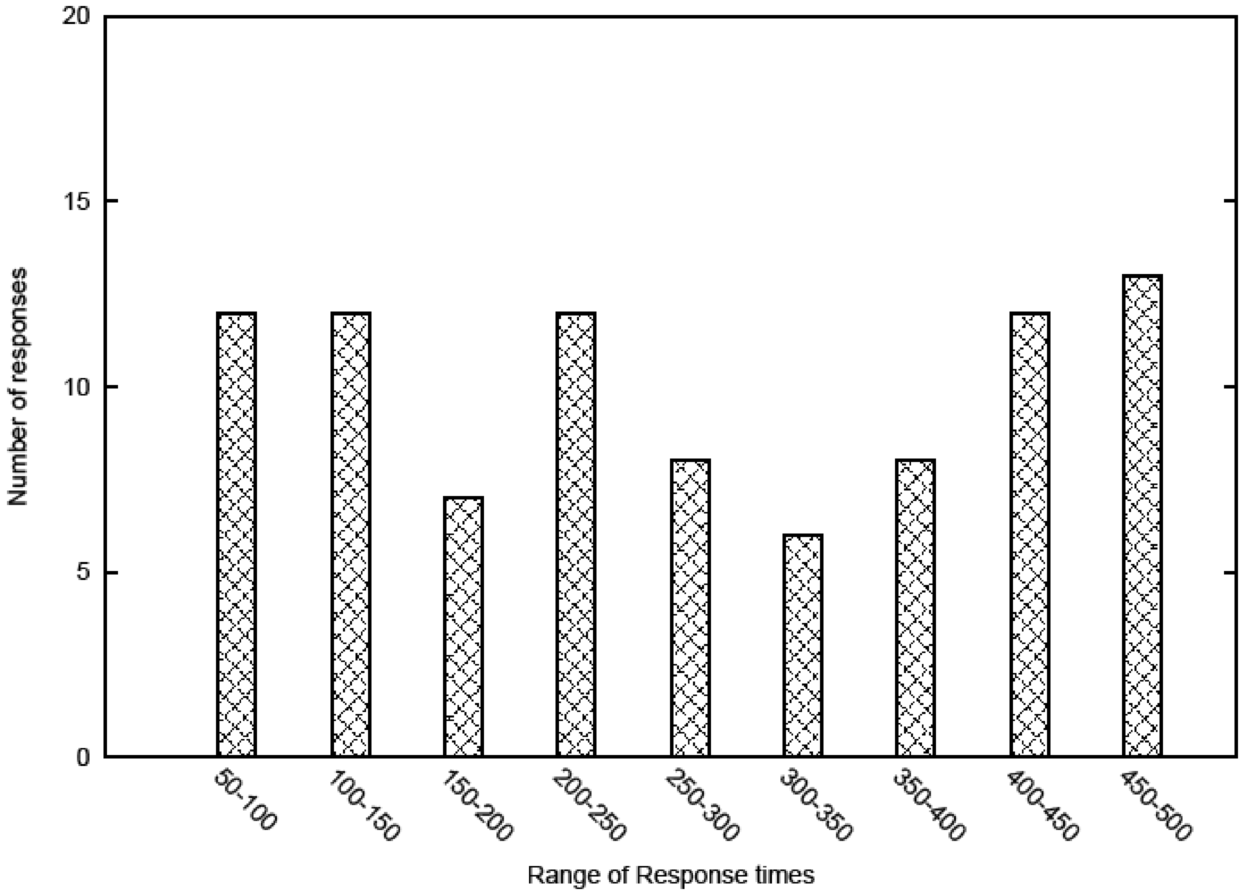
LRED(1) = ~LDF
LRED(inf) = EDF

LRED Algorithm

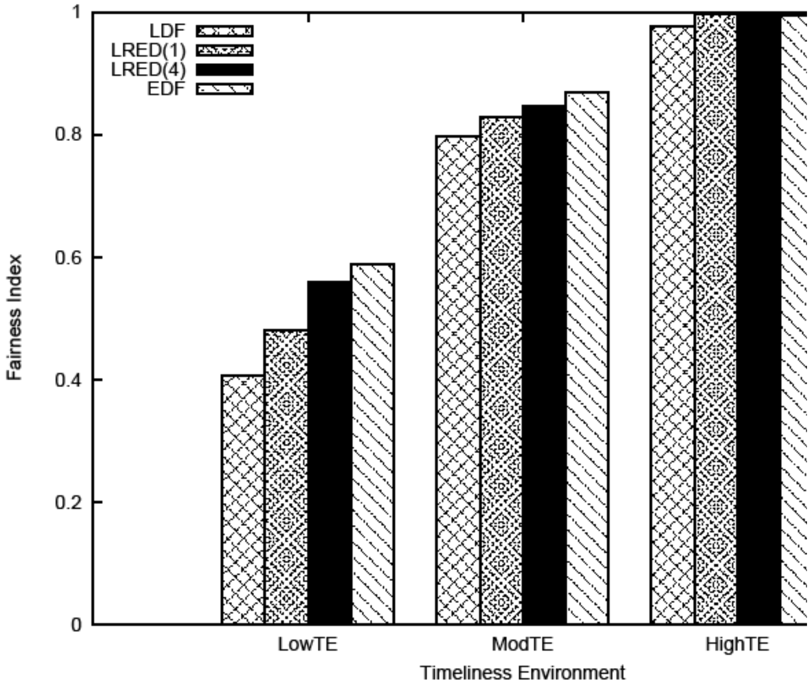
Algorithm 1 LRED(n)

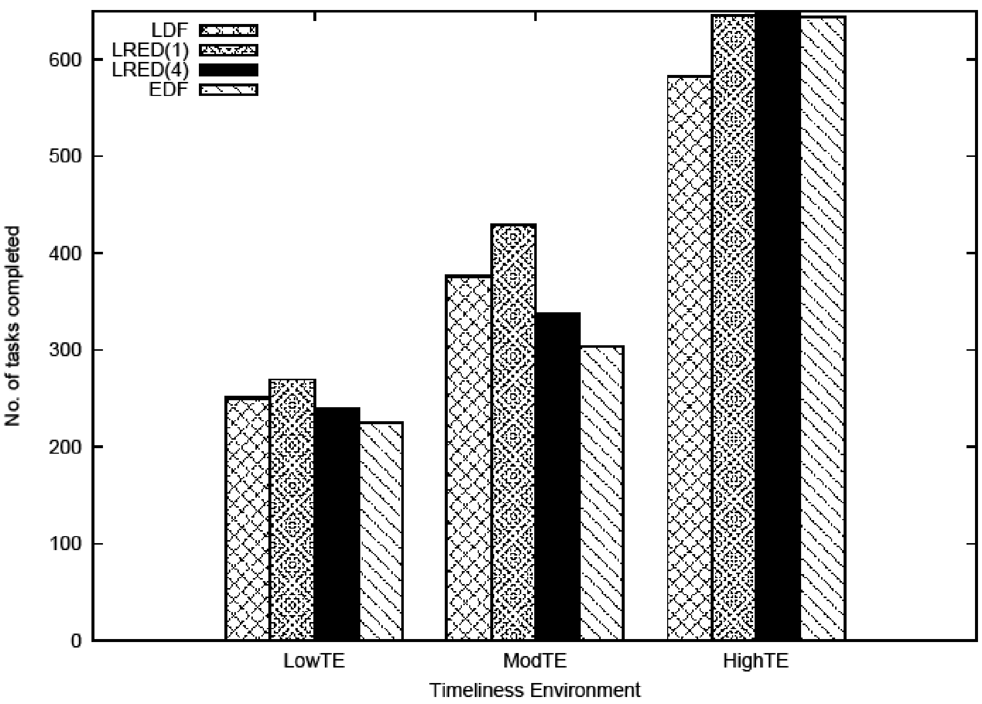
```
W ← Set of all available workers
Sort W in increasing order of τ
Sort the task pool in increasing order of D
while W is non-empty do
    Organize the task pool into the list {S_1, S_2, ..., S_max} based
    on τ of workers in W
    V ← Set of all tasks in the list {S_n, ..., S_2, S_1}
    if V is non-empty then
        T ← First task from the first non-empty set S_k in V
        Schedule T to k most timely workers
        Update W by removing the k assigned workers
    else if n < max then
        LRED(n+1)
    else
        break
    end if
end while
```

PL Trace


20
15
10
5
0
Number of responses
50-100
100-150
150-200
200-250
250-300
300-350
350-400
400-450
450-500
Range of Response times

Results





TSR = .9, lowTE [80-150], modTE [120-200], highTE [200-350]

Result Summary

- Smaller n for LRED provides better throughput
- Larger n for LRED produces higher fairness
- Can be tuned
- The throughput-fairness tradeoff
  - is more visible than at high loads
  - becomes more significant as the overall timeliness level of the environment decreases

Questions?

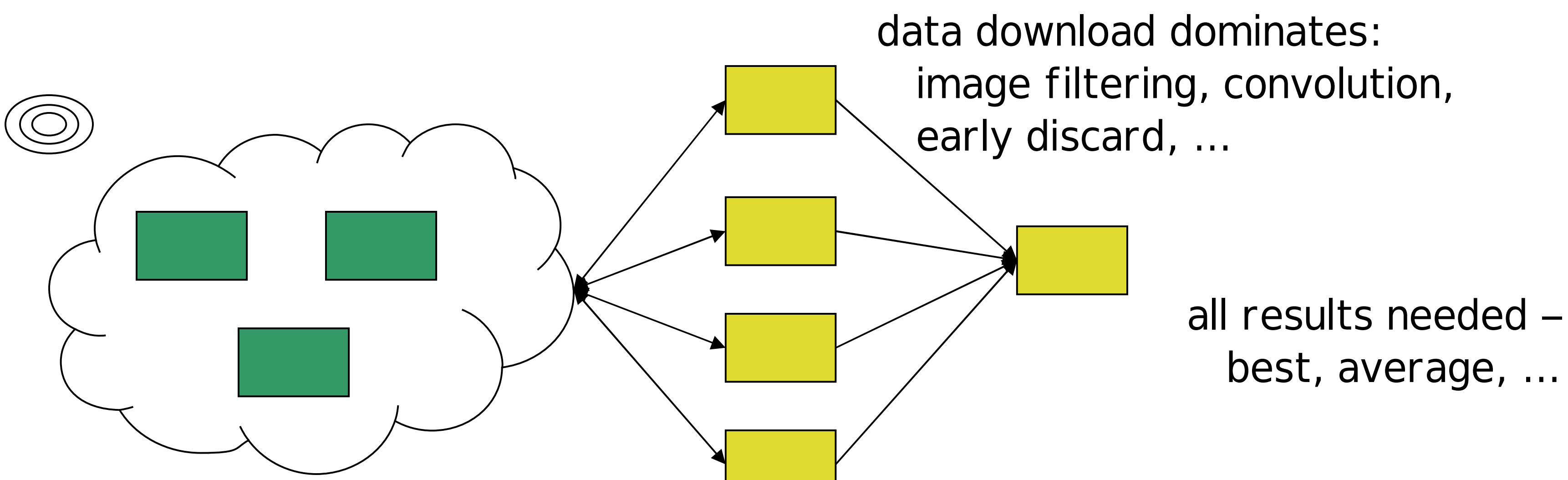
Dynamic Communication

- Dynamically selecting data servers
- Dynamically estimating bandwidth
- Dynamic workflow optimization

- All in the context of Grid applications

Dynamically selecting data servers

- Nodes download data from replicated data nodes
  - Nodes choose "data servers" independently (decentralized)
  - Minimize the maximum download time for all worker nodes (*communication makespan*)

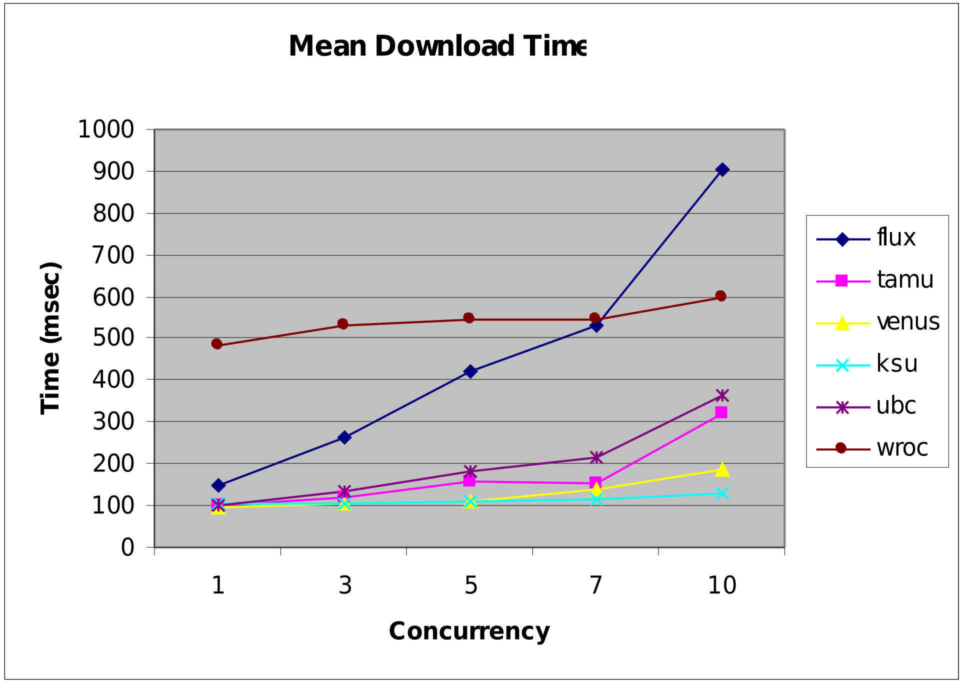


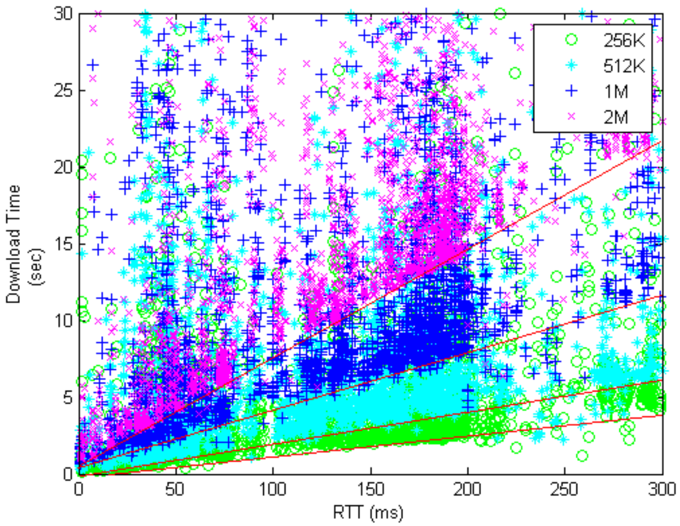
Motivation

- Many distributed network applications require massive data
  - HEP, image analysis, ...
  - Such datasets are often replicated to data servers
- Challenges in data delivery
  - Proximity of clients to data servers
  - Unreliability and heterogeneity of data servers
  - Load balancing

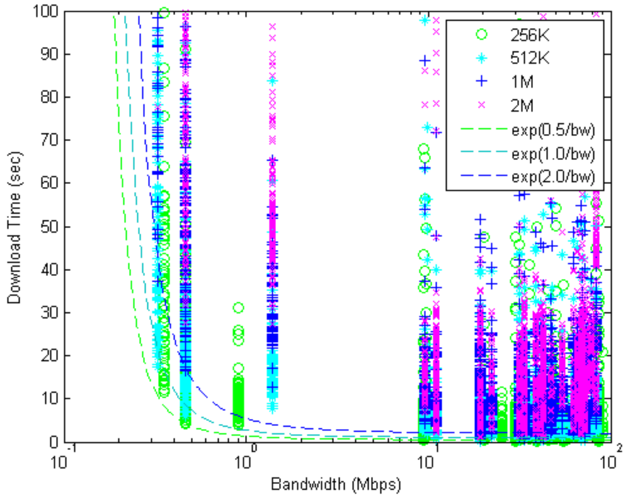
Data node selection

- Several possible factors
  - Proximity (RTT)
  - Network bandwidth
  - Server capacity





[Download Time vs. RTT - linear]



[Download Time vs. Bandwidth - exp]

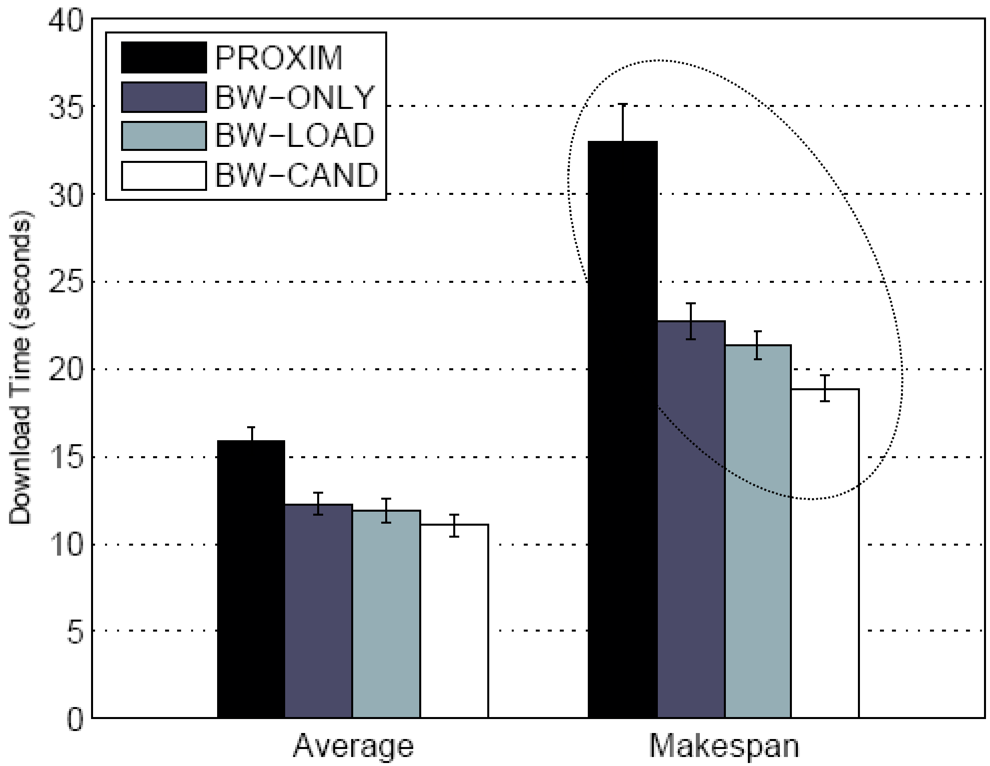
Server Selection Strategy

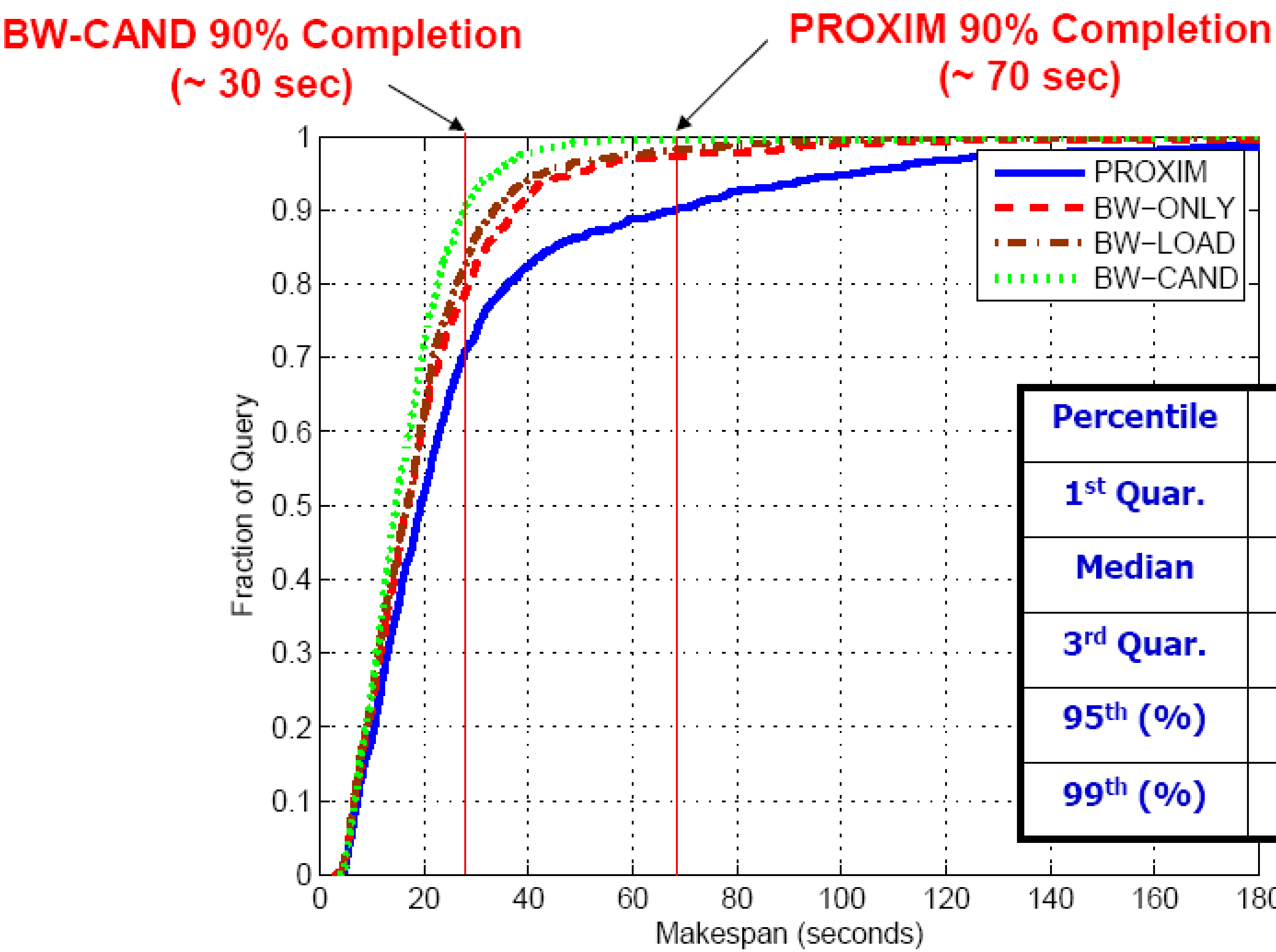
- Based on observations
  - Servers with *low bandwidth* (e.g. < 1Mbps) are to be avoided
    - even if RTT is small
  - Servers with *high bandwidth* (e.g. > 10Mbps) are to be favored
    - use RTT as a discriminator
  - Servers with *medium bandwidth* (e.g. 1-10Mbps) are best discriminated by system load

Heuristic Ranking Function

- Query to get candidates, do RTT/bw probes
- Node i, data server node j
  - Cost function = $rtt_{i,j} * exp(k_j / bw_{i,j})$, k_j load/capacity
- Least cost data node selected independently
- Three server selection heuristics that use k_j
  - BW-ONLY: $k_j = 1$
  - BW-LOAD: k_j = *n-minute average load (**past**)*
  - BW-CAND: k_j = *# of candidate responses in last m seconds (~ **future** load)*

Performance Comparison

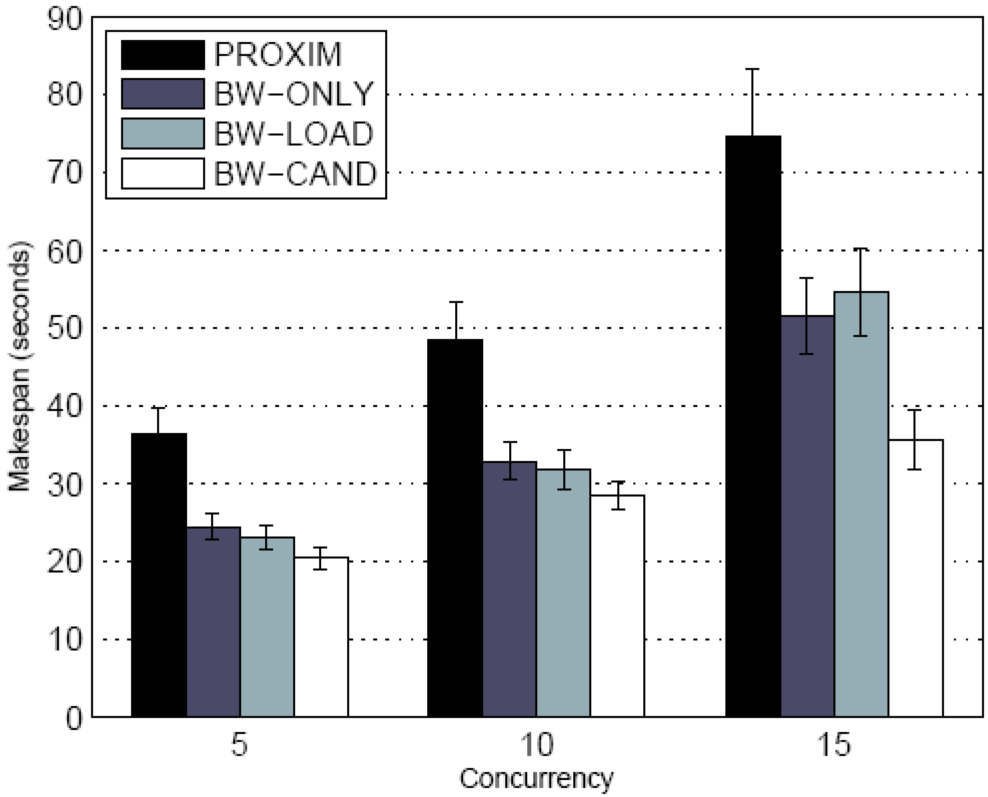


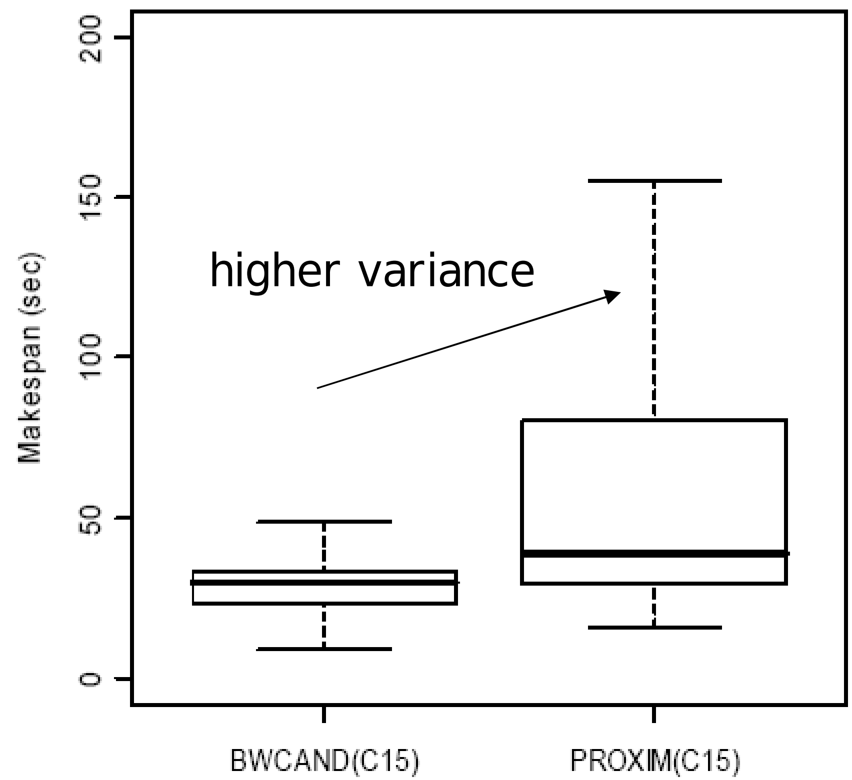


(Unit: second)

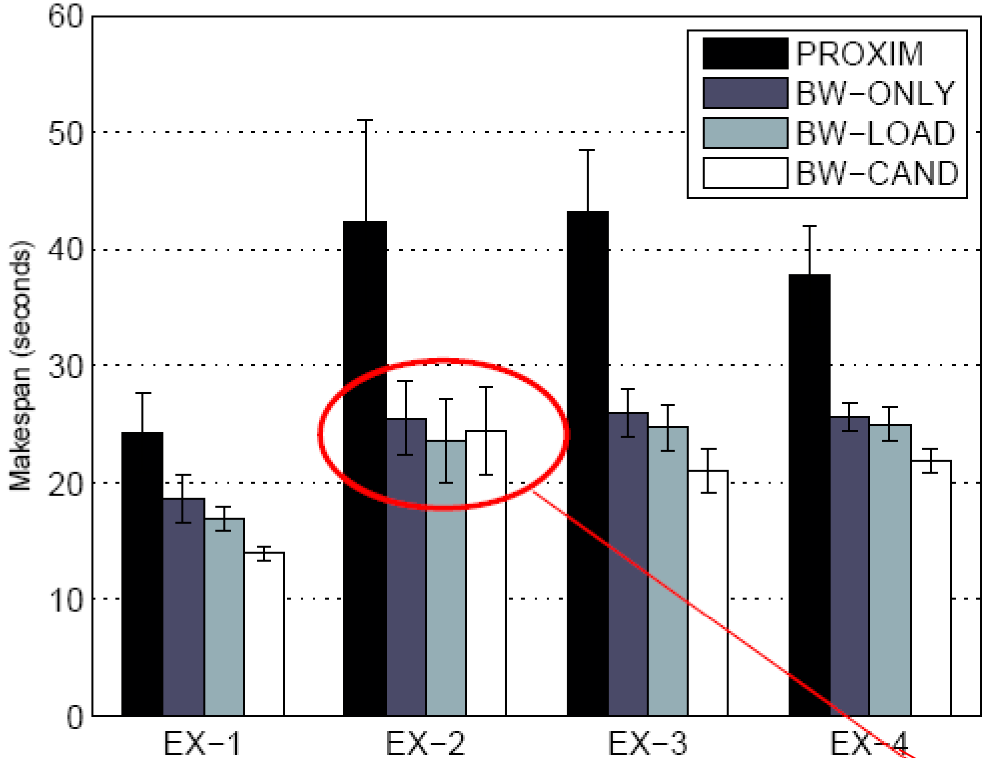
Percentile	PROXIM	BW-CAND
1st Quar.	13.3	10.9
Median	21.9	16.3
3rd Quar.	33. 0	23.2
95th (%)	95.4	36.4
99th (%)	192.6	59.1

Impact of Loading





Comparison of Individual Experiments



Different server configs

Table 2. Server Bandwidth Distribution

Class	Low $< 1Mbps$	Medium $1 - 10Mbps$	High $> 10Mbps$
EX-1	5%	26%	67%
EX-2	12%	6%	82%
EX-3	0%	24%	76%
EX-4	0%	24%	76%

Take away

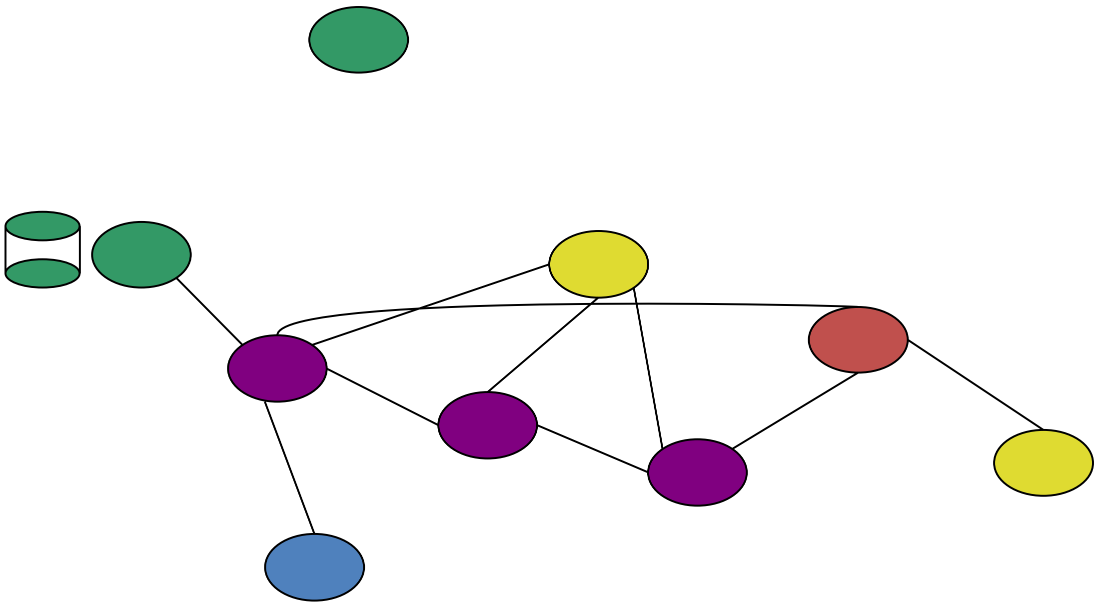
- Bandwidth, proximity, load, and capacity all matter!
- Both heterogeneity and load balancing must be taken into account
- But reliance on RTT and BW measurements

Dynamically Estimating Bandwidth

- Bandwidth, RTT estimation can be applied to:
  - prior situation (many workers, many data servers)
- Compute-oriented applications (e.g BLAST)
  - similar to prior situation (many workers, many data servers) although communication is less of a b-neck
- Worker selection for data-intensive tasks
  - where to place workers?

Dynamically Estimating Bandwidth

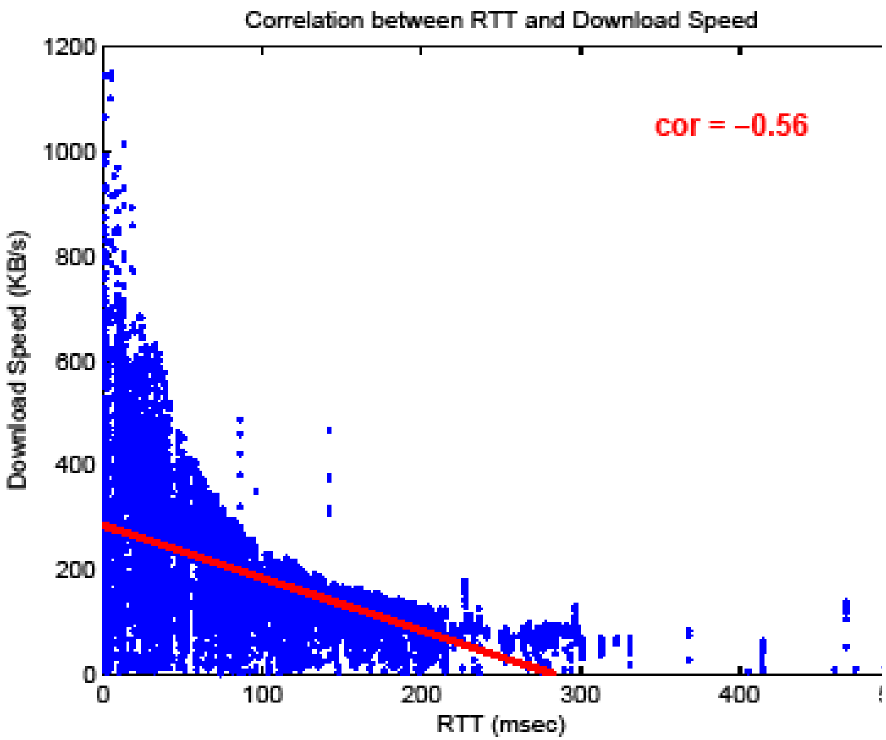
- Data-intensive computation needs access to one or more data sources – data may be very large
  - bw probes to too expensive

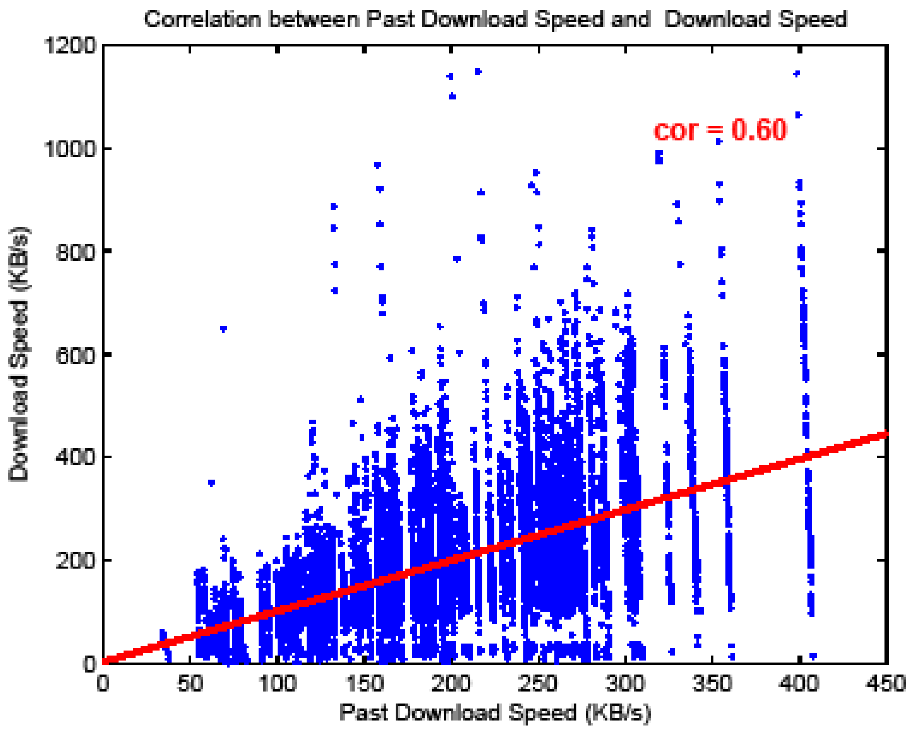


The Problem

- Where to run a data-intensive computation (or place a worker)?
  - from a set of candidates
- Unlikely a candidate knows downstream bw from particular data nodes
- Idea: infer bw from prior observations or from neighbor observations w/r to data nodes!
- Simplify presentation: single worker, no replication, single data object

Some Observations

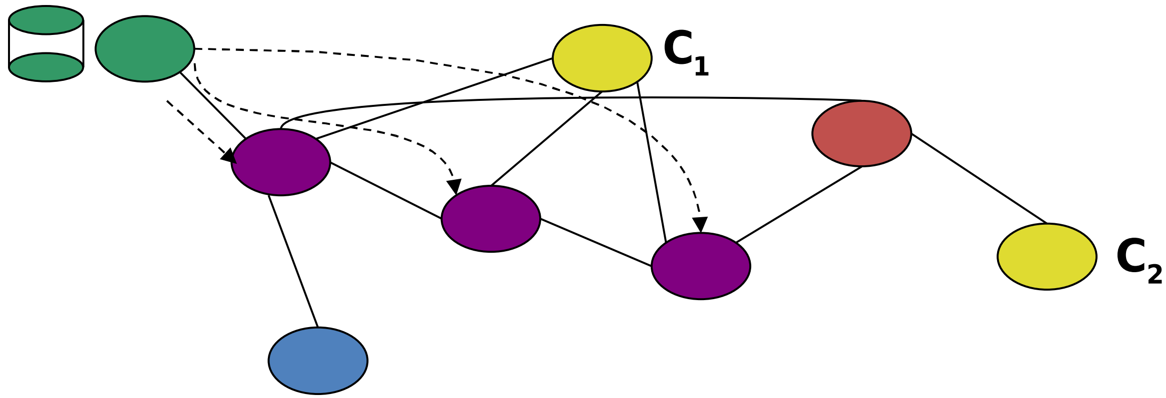




RTT and Download Speed

Past Download Speed to Download Speed

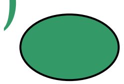
Estimation Technique



- C_1 may have had little past interaction with
  - … but its neighbors may have
- For each neighbor generate a download estimate:
  - *DT*: prior download time to from neighbor
  - *RTT*: from candidate and neighbor to respectively
  - *DP*: average weighted measure of prior download times for any node to any data source

Estimation Technique (cont'd)

- Download Power (*DP*) characterizes download capability of a node
  - DP = average (*DT* * *RTT*)
  - DT not enough (far-away vs. nearby data source)

- Estimation associated with each neighbor n_i
  - $ElapsedEst[n_i] = \alpha \cdot \beta \cdot DT$
    - α : my_RTT/neighbor_RTT (to )
    - β : neighbor_DP /my_DP
    - no active probes: historical data, RTT inference
- Combining neighbor estimates
  - mean, median, min, ...
  - median worked the best
- Take a min over all candidate estimates

NEIGHBOR

$$DP_h = \frac{1}{|\mathcal{H}_h|} \sum_{i=1}^{|\mathcal{H}_h|} \left(\frac{\mathcal{H}_h^i.size}{\mathcal{H}_h^i.elapse} \times \mathcal{H}_h^i.distance \right)$$

$$NeighborEstim_h(n, o) = \alpha \cdot \beta \cdot elapse_n(o)$$

where

$$\alpha = \frac{DP_n}{DP_h}, \beta = \frac{distance_h(server(o))}{distance_n(server(o))},$$

$$NeighborEstim_h(o) = median_{n_i \in N}(NeighborEstim_h(n_i, o))$$

SELF

$$\overline{Distance_h} = \frac{1}{|\mathcal{H}_h|} \cdot \sum_{i=1}^{|\mathcal{H}_h|} \mathcal{H}_h^i.distance$$

$$\overline{DownSpeed_h} = \frac{1}{|\mathcal{H}_h|} \cdot \sum_{i=1}^{|\mathcal{H}_h|} \frac{\mathcal{H}_h^i.size}{\mathcal{H}_h^i.elapse}$$

$$SelfEstim_h(o) = \delta \cdot \frac{size(o)}{\overline{DownSpeed_h}}$$

where

$$\delta = \frac{distance_h(server(o))}{\overline{Distance_h}}$$

← must be an active probe

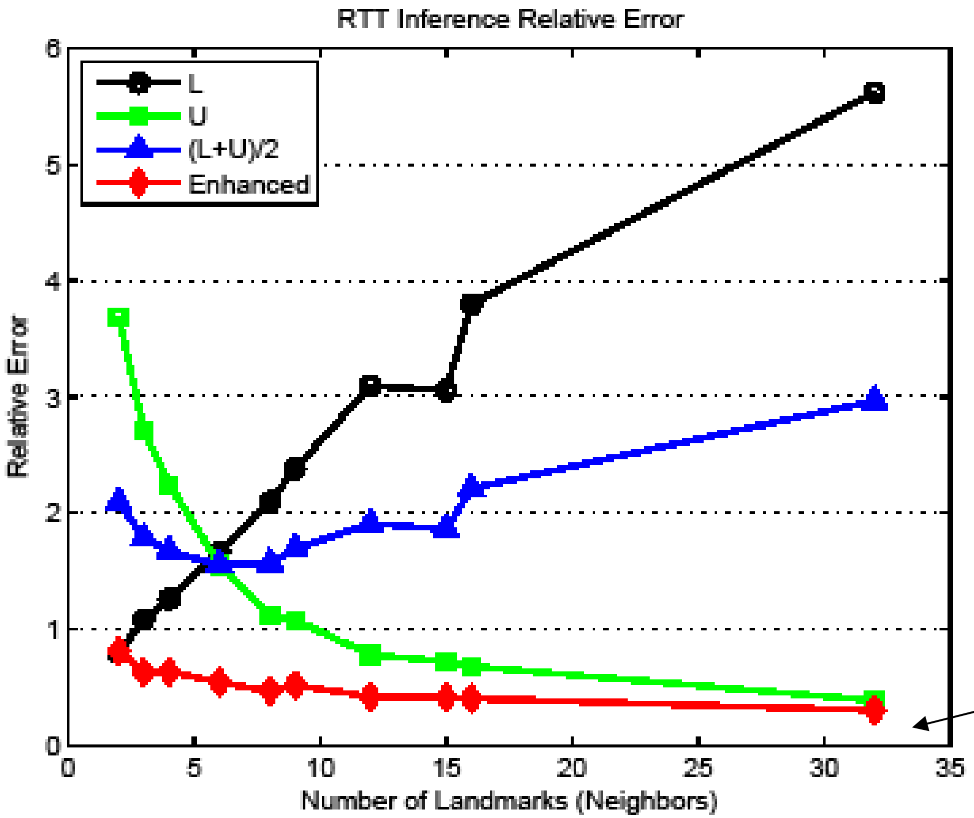
this is an issue with 1000s of candidates ...

RTT Inference

We can do inference if our neighbors have an RTT to the server!
Triangle inequality:

latency (a, c) lies between

$$|latency(a,b) - latency(b,c)| \text{ and } latency(a,b) + latency(b,c)$$

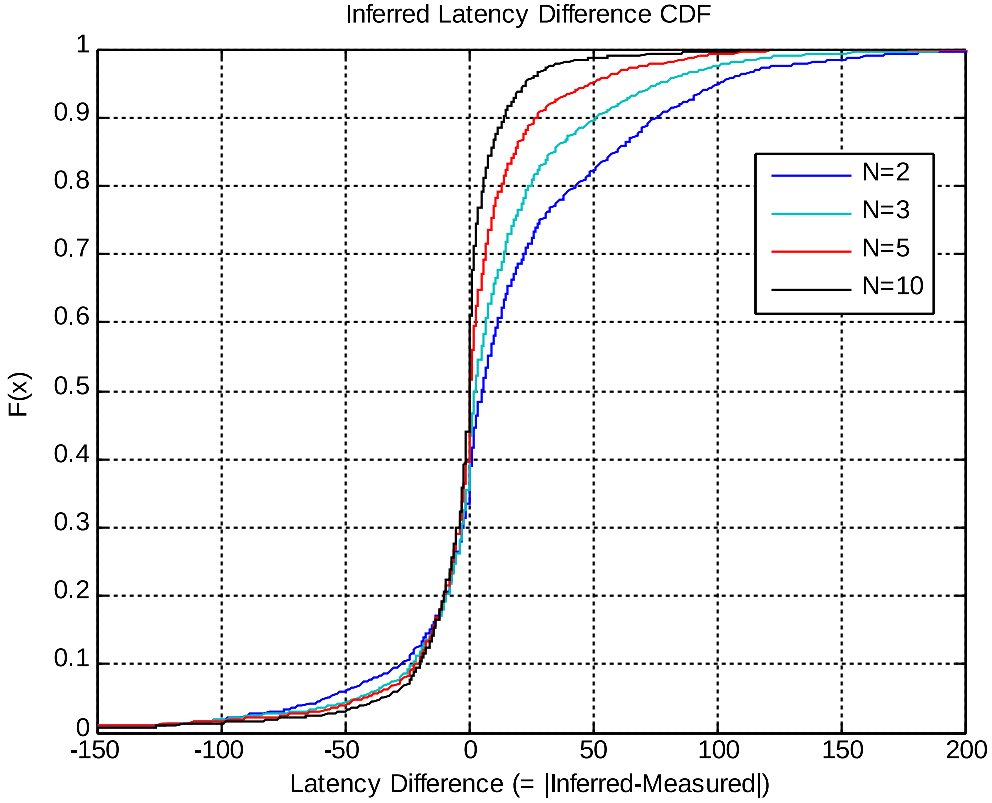


<u>Idea:</u> remove observations that violate the triangle inequality

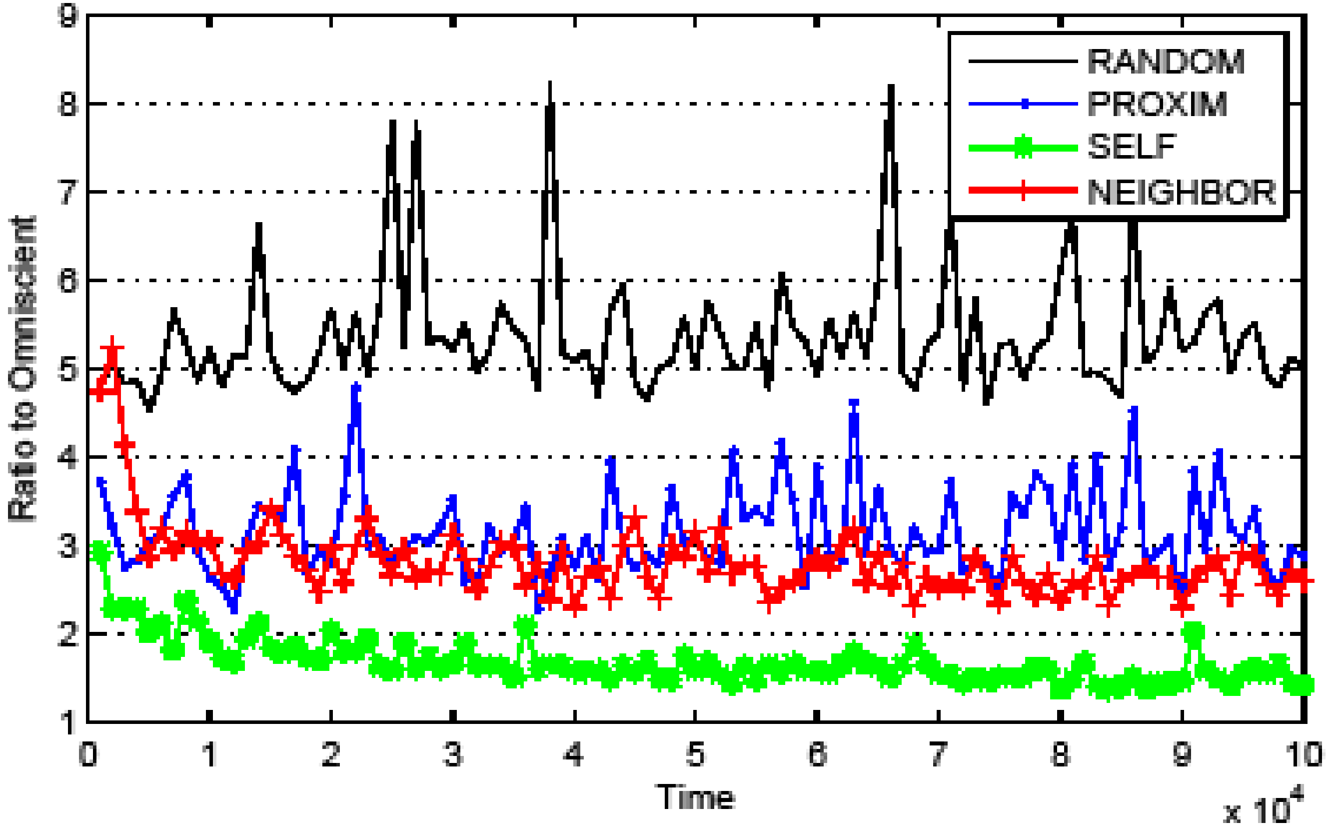
ours

RTT Inference Result

- More neighbors, greater accuracy
- With 5 neighbors, 85% of the estimations < 16% error

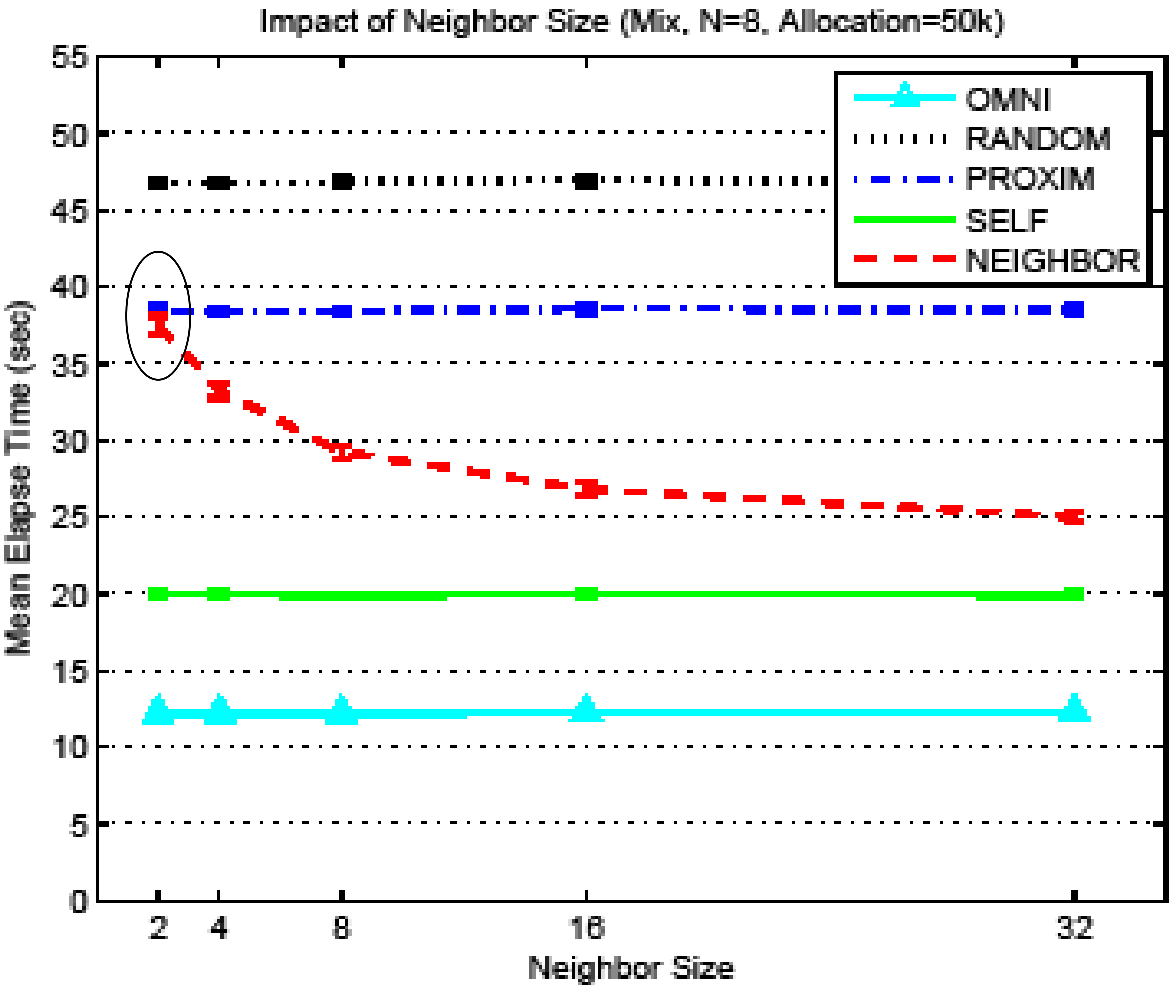


Results

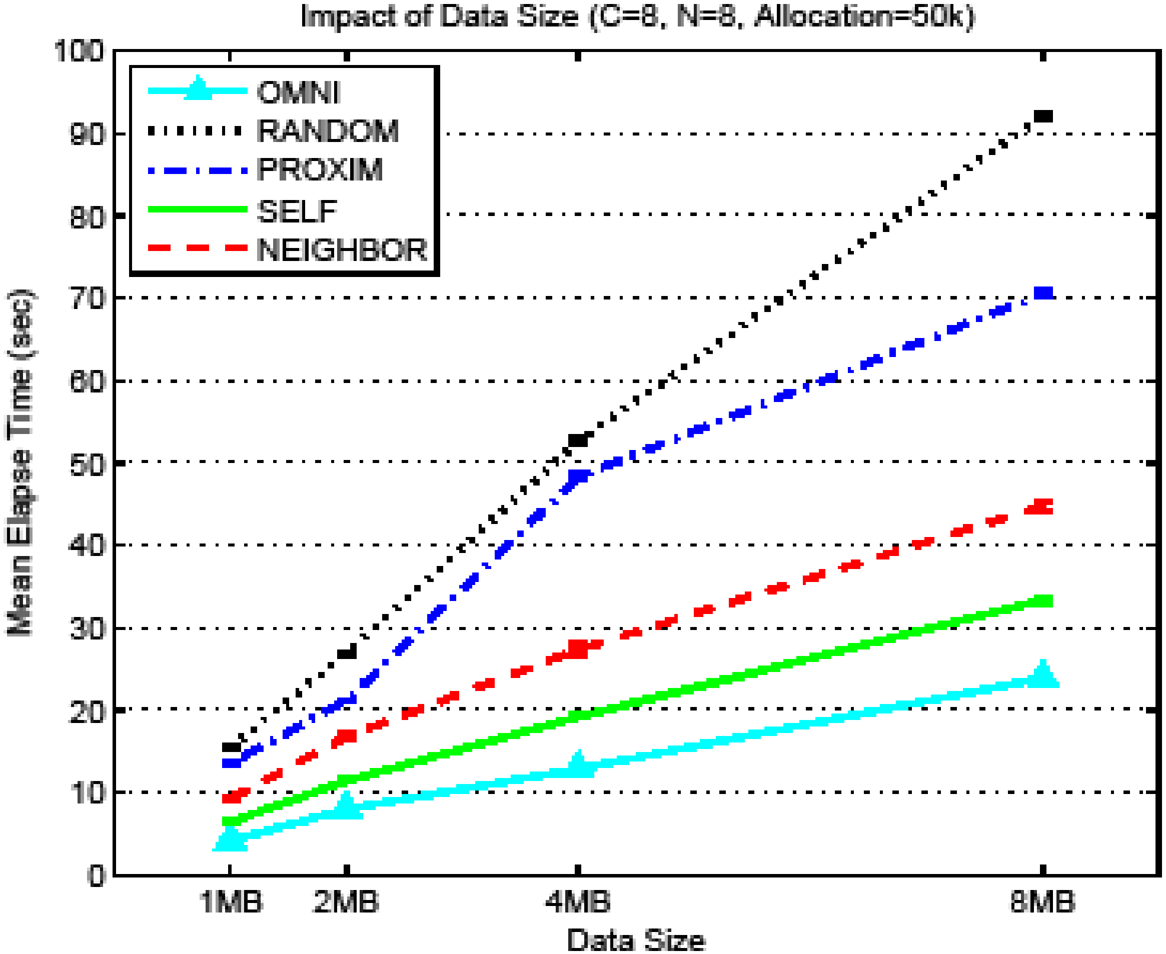


8 neighbors

Neighbor size

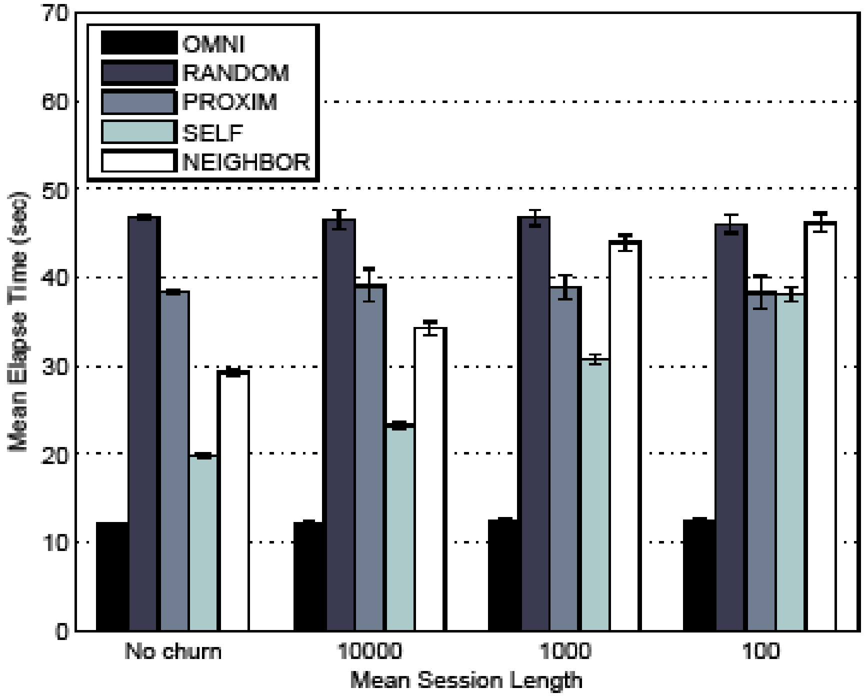


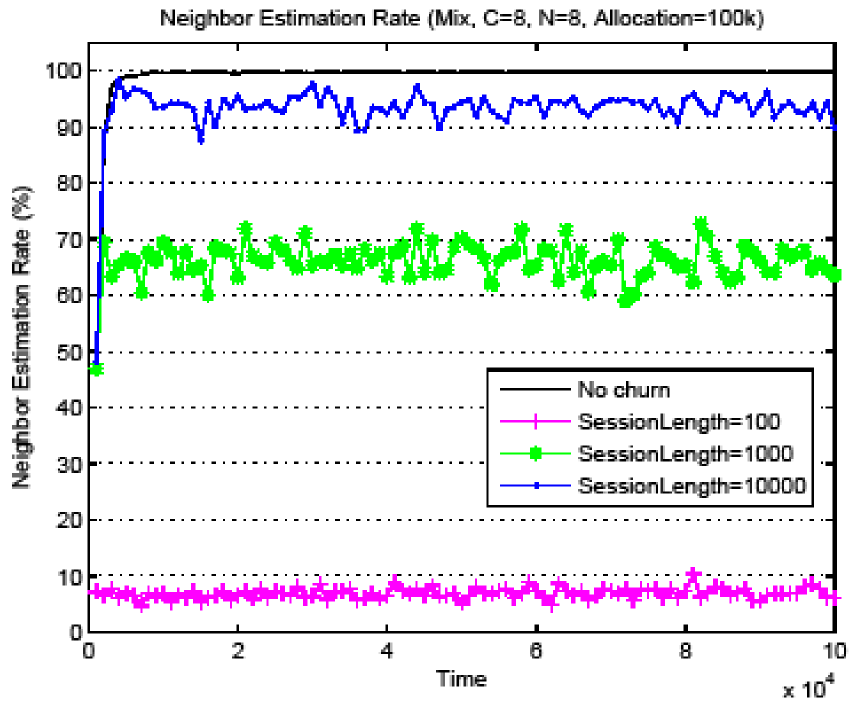
Data size



Impact of Data Size (C=8, N=8, Allocation=50k)
OMNI
RANDOM
PROXIM
SELF
NEIGHBOR
Mean Elapse Time (sec)
0
10
20
30
40
50
60
70
80
90
100
1MB
2MB
4MB
8MB
Data Size

Churn


OMNI
RANDOM
PROXIM
SELF
NEIGHBOR
Mean Elapse Time (sec)
0
10
20
30
40
50
60
70
No churn
10000
1000
100
Mean Session Length


Neighbor Estimation Rate (Mix, C=8, N=8, Allocation=100k)
Neighbor Estimation Rate (%)
0
10
20
30
40
50
60
70
80
90
100
No churn
SessionLength=100
SessionLength=1000
SessionLength=10000
0
2
4
6
8
10
Time
x 10^4

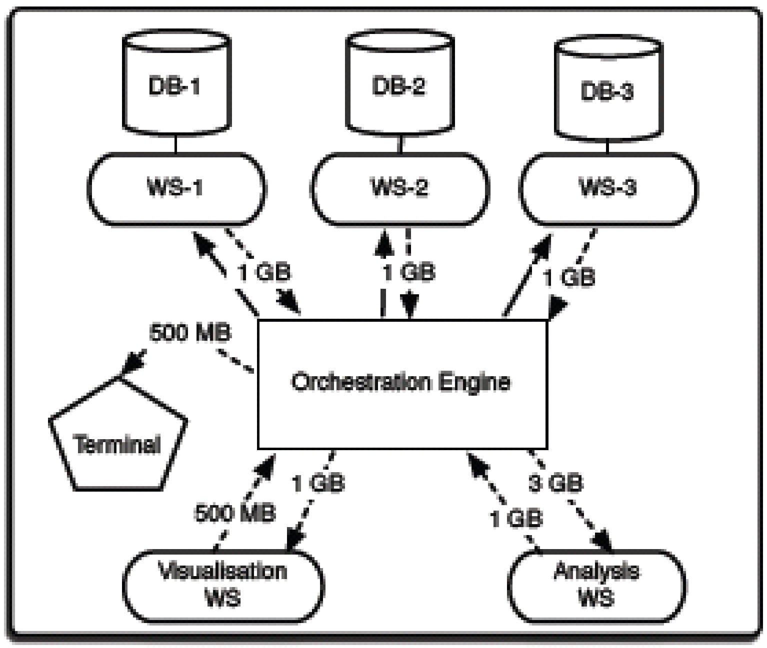
Take Away

- Locality between a data source and a node
  - scalable, no (or minimal) probing needed
  - allows for ranking based on bw estimates, provides RTT

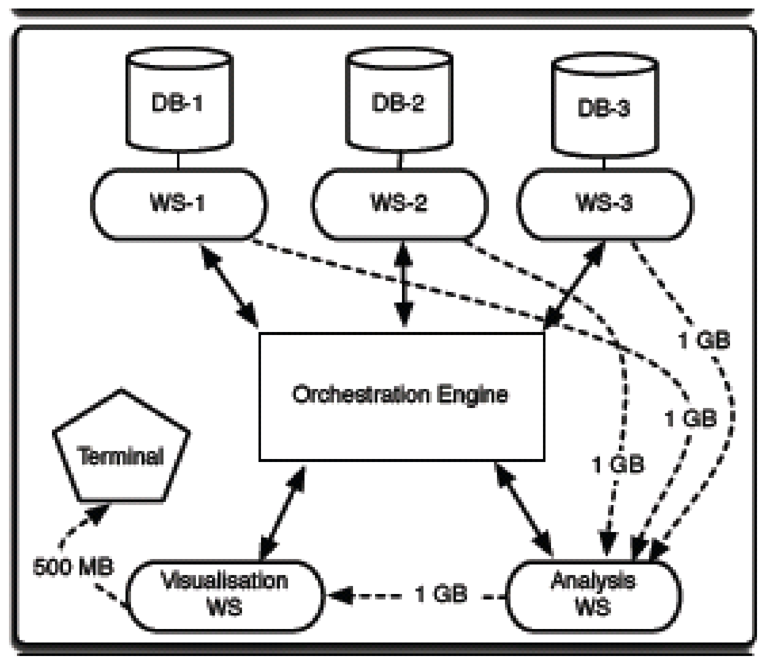
- Wide application
  - useful in many scenarios based on ranking choices

Data flow optimization

- Workflow architectures

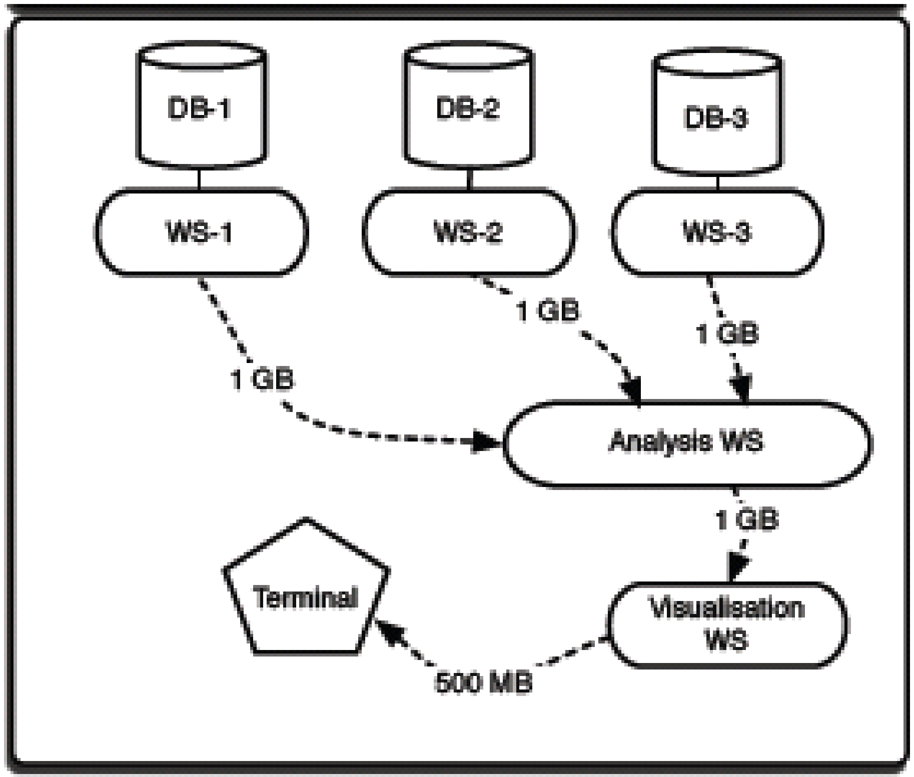


centralized data and control



distributed data and central control

Choreography

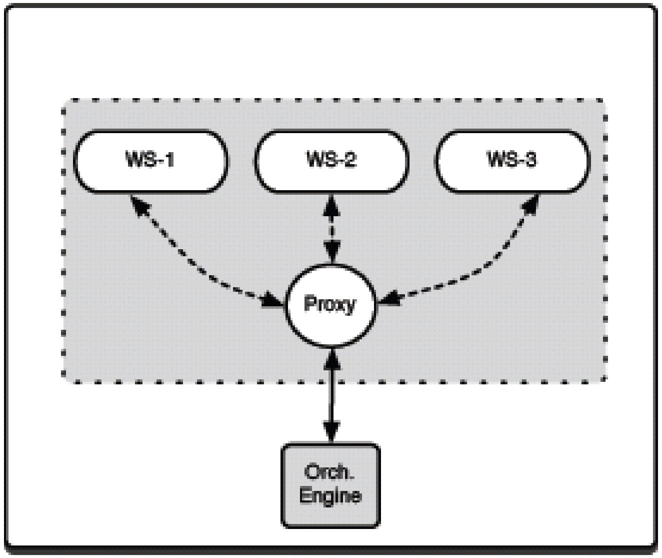


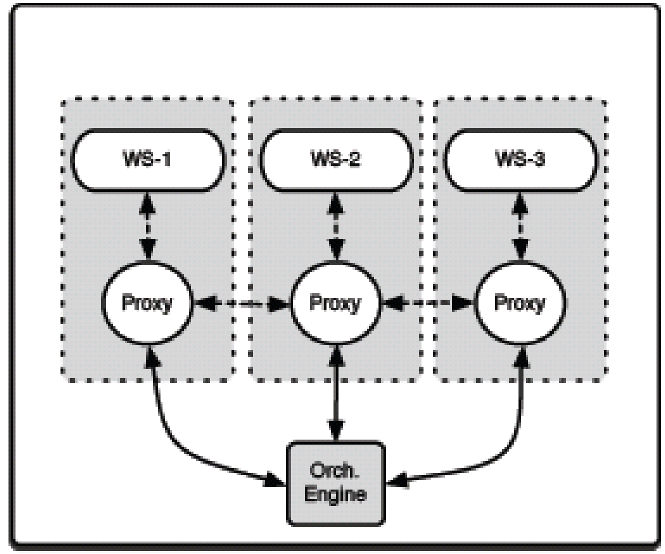
distributed data and distributed control

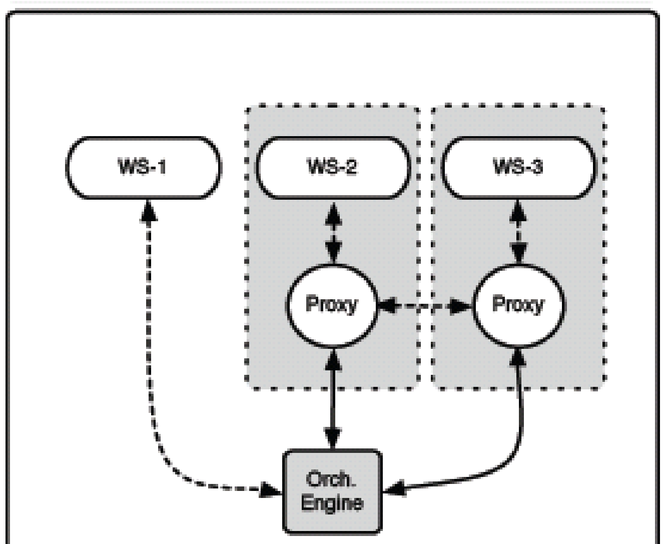
Comparisons

- Pure Choreography
  - More collaborative
  - WS-CDL exists but no implementations
  - Best possible performance
  - Requires users modify their services
- Pure Orchestration
  - Control and data-flow managed by a central engine
  - Worst possible performance
  - No service modification required

Middle-Ground Solution







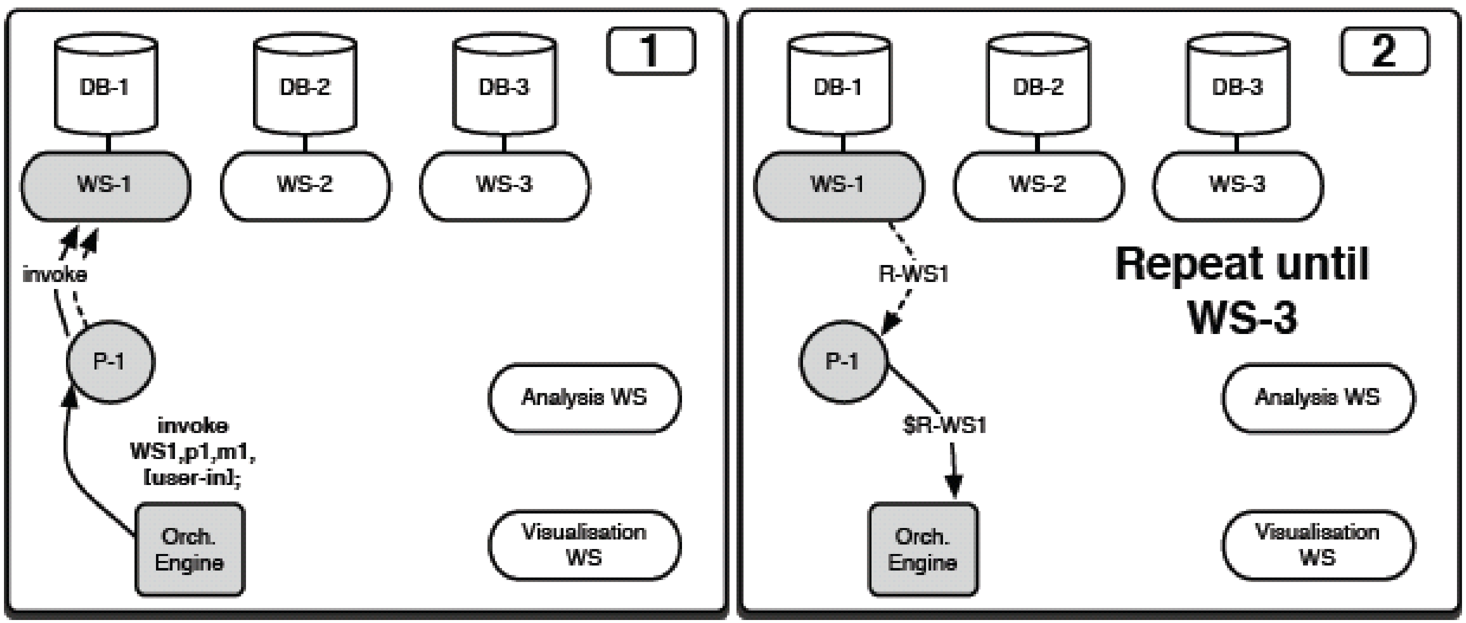
proxy is a WS; located near the service

Proxy API

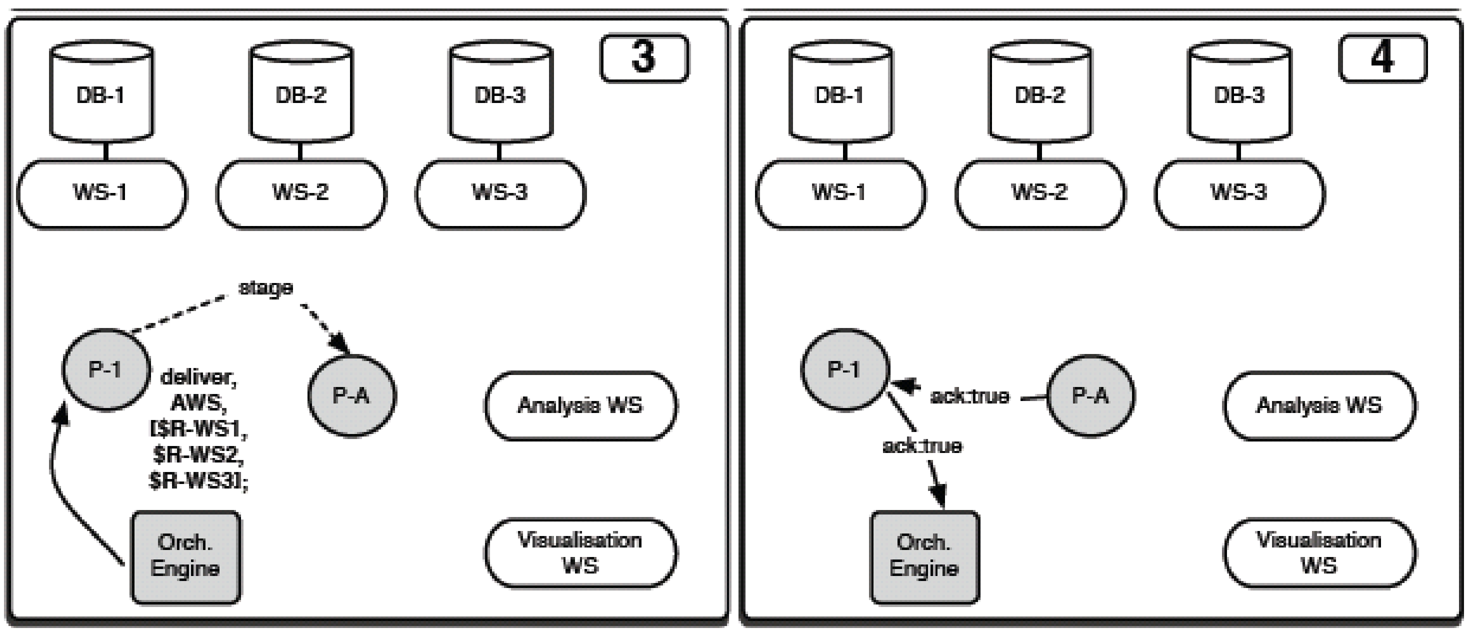
```
public interface proxy {
   //Proxy CORE methods
   public String invoke(String wsdl, String port, String op_name, Object[] params)
           throws InvocationParameterError, VariableNotFoundError, ServiceInvocationError;
   public boolean deliver(String proxy_wsdl, String[] dataToMove)
           throws VariableNotFoundError, ServiceInvocationError;
   public boolean stage(Hashtable dataToMove)
           throws ServiceInvocationError;
   public Object[] returnData(String[] dataToReturn)
           throws VariableNotFoundError;
   public boolean flushTempData(String [] dataToRemove)
           throws VariableNotFoundError;

   //Proxy ADMIN methods
   public void addService(String wsdl)
           throws ProxyAdminError;
   public void removeService(String wsdl)
           throws VariableNotFoundError;
   public String[] listOperations(String wsdl, String port)
           throws VariableNotFoundError;
   public String[] listOpParameters(String wsdl, String port, String op_name)
           throws VariableNotFoundError;
   public String[] listOpReturnType(String wsdl, String port, String op_name)
           throws VariableNotFoundError;
   public String[] listServices();
}
```

Example

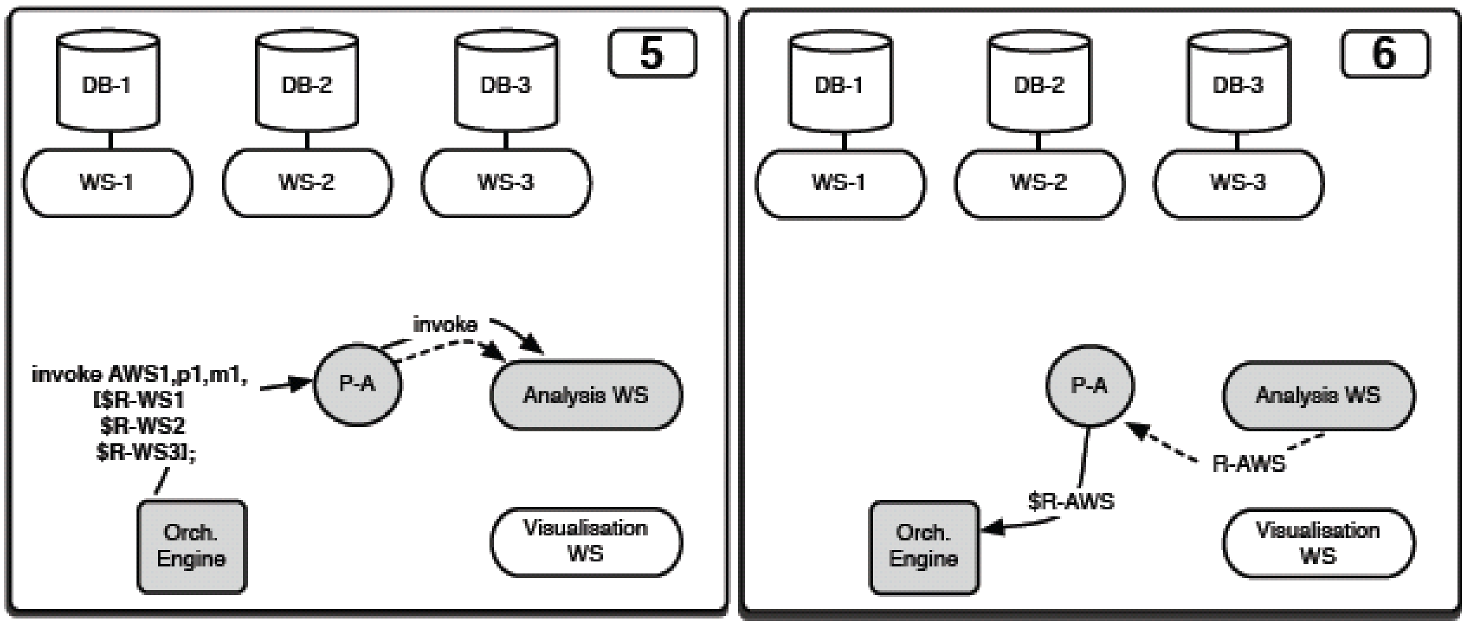
Back to our earlier example



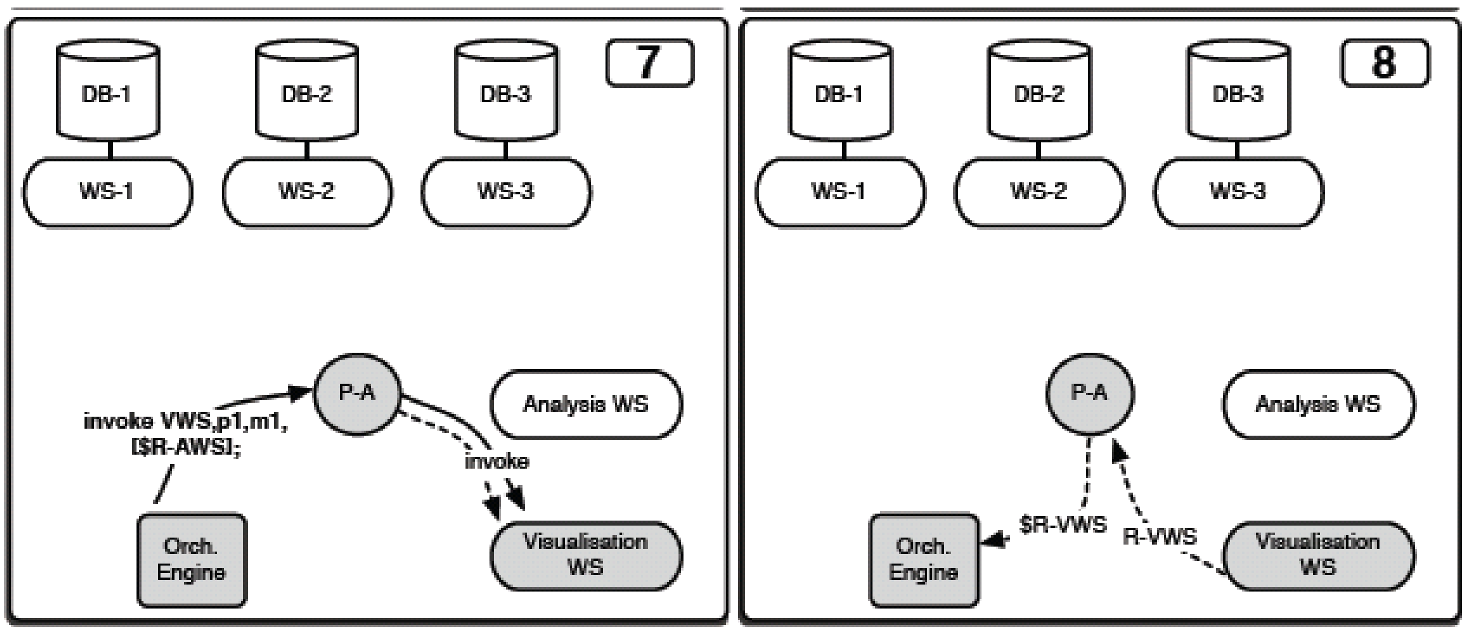
Example (cont'd)



Example (cont'd)



Example (cont'd)



Issues

- Nice research questions
  - Proxy selection
  - Proxy assignment
  - Proximity
  - Load balance
- Currently exploring these on PlanetLab

Questions?

Dynamic Metrics

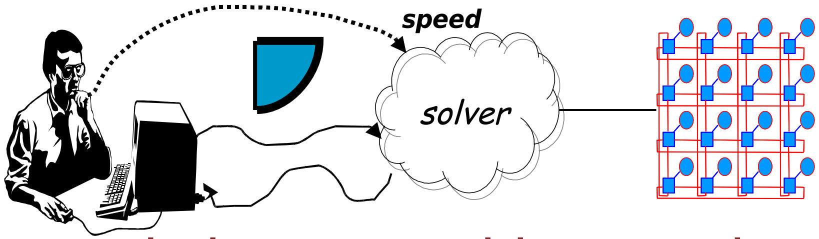
- How to characterize and measure the behavior of dynamic (Grid) systems?
- Current metrics are absolute and therefore inadequate
- Dynamism suggests we need 'first derivative" metrics

Metrics Today

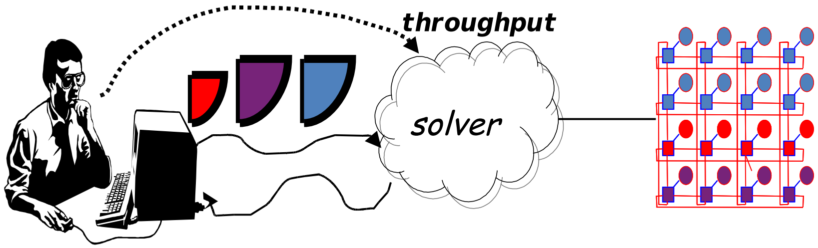
- Typical performance metrics
  - exploit and address the common case
  - if application X is given Y resources, then Perf(X, Y) can be expected
  - F= min/max { **avg** [Perf (X, Y)] }

- Average performance has always been the metric to measure success
  - completion time (minimize)
  - throughput (maximize)

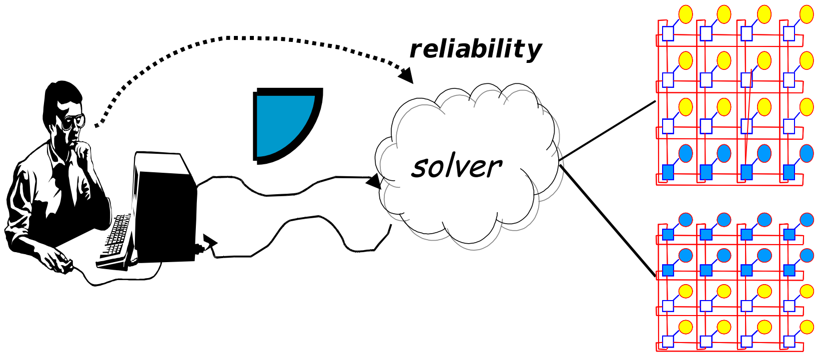
Traditional Metrics



user is interested in speed only for each request



user is interested in throughput



user is interested in reliability

Another metric

- Other metrics may be useful for dynamic (Grid) systems
  - Average or common case may be less important

- Account for dynamic characteristics
  - resource availability is stochastic
  - demand is stochastic
  - execution time is stochastic

- Robustness
  - performance robustness may be as (or more) important than "high performance"

Scheduling for Robustness

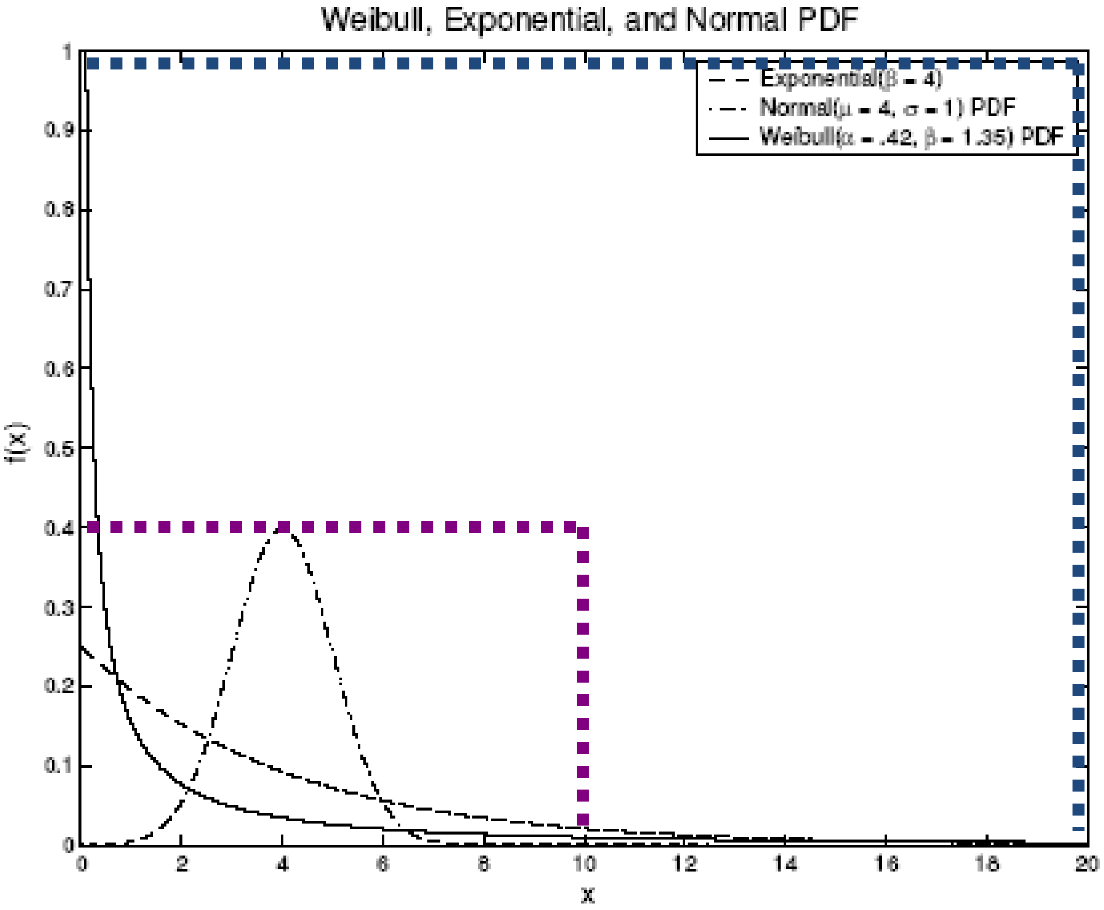
"Robustness is the **persistence** of certain specified system **features** despite **perturbations** being applied to the system"

- Performance robustness for applications
  - small perturbations result in large fluctuations in behavior
  - perturbations are inherently unpredictable
  - **"compressed" variance, yet good performance**
- Perturbations
  - arrival patterns (demand)
  - resource availability
  - **execution times (data-dependent)**

Application execution time

- Application execution times are stochastic
- "Heavy-tailed" scientific computing application server
  - irregular search that results from optimization problems
    - branch-and-bound
  - highly data- and control-dependent execution times
    - solve my NxN system using Jacobi … or CG …
  - occasional large request can dominate the workload => this is the perturbation
  - large variance

Stochastic execution times



Scheduling for Robustness

- Example 1
  - application contains **two** types of tasks
  - one normally distributed (N), the other heavy-tailed (H)
  - **two** machines can run either type of task
  - application instance has **three** tasks
  - one of type N (T_1) and two of type to H (T_2 and T_3)

- Scheduling
  - policy 1: balance based on expected values
  - policy 2: balance based on 99^{th} percentile

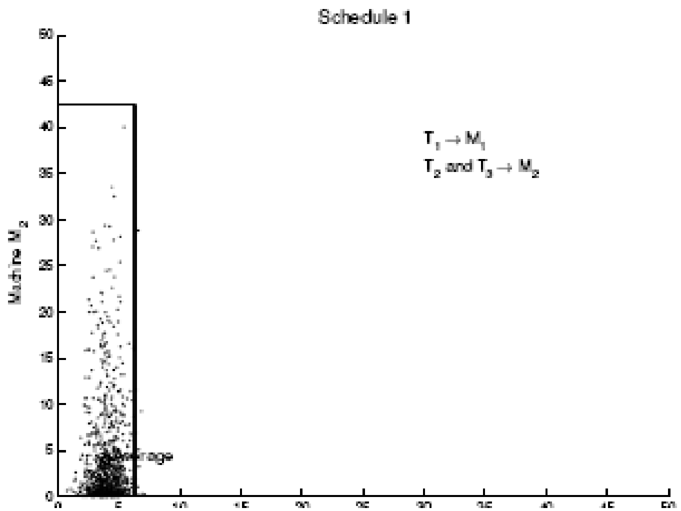
Task	Execution Times	Mean	$\{x \mid P(X < x) = .99\}$
T_1	~Normal(4,1)	4	6.33
T_2	~Weibull(0.5, 1)	2	21.2
T_3	~Weibull(0.5, 1)	2	21.2

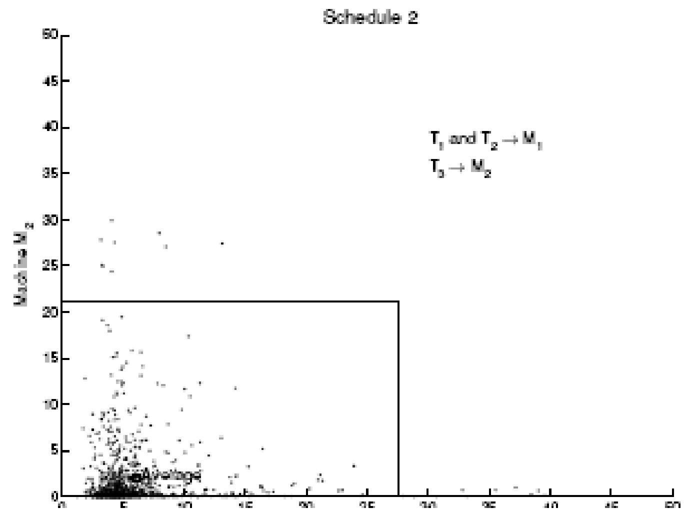
	Schedule	Machine M_1	Machine M_2
policy 1 =>	S_1	T_1	T_2,T_3
policy 2 =>	S_2	T_1,T_2	T_3

Scheduling for Robustness (cont'd)

- Robustness as a function of <u>makespan</u>

M2





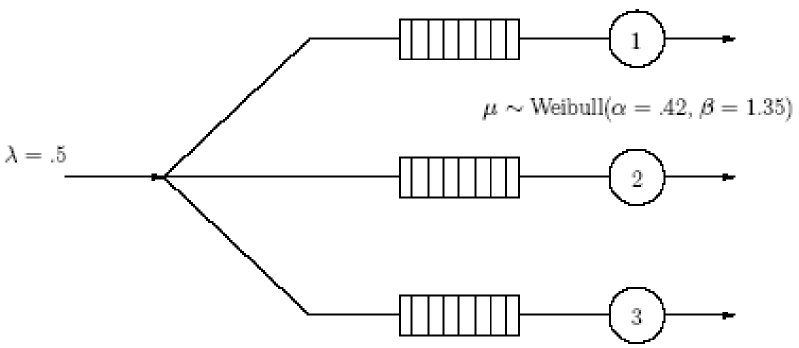
M1

	Average Case	Extreme Case	range
Schedule S_1	make-span = 4	make-span = 42.5	38.5
Schedule S_2	make-span = 6	make-span = 27.5	21.5
Benefit/Penalty	2	15	

- Robustness as a function of <u>completion time</u>

Demand Example

- Example 2
  - independent requests to a service
  - heavy-tailed
  - three machines



- Scheduling
  - policy 1: shortest queue
  - policy 2: send to machine with most recently started execution (avoid when same req. running for a while)
    - exploits heavy-tailed property that running time is a good indicator of future remaining running time

- Robustness as a function of <u>waiting time</u>

Demand Example (cont'd)

Queue statistics for 10,000 observations

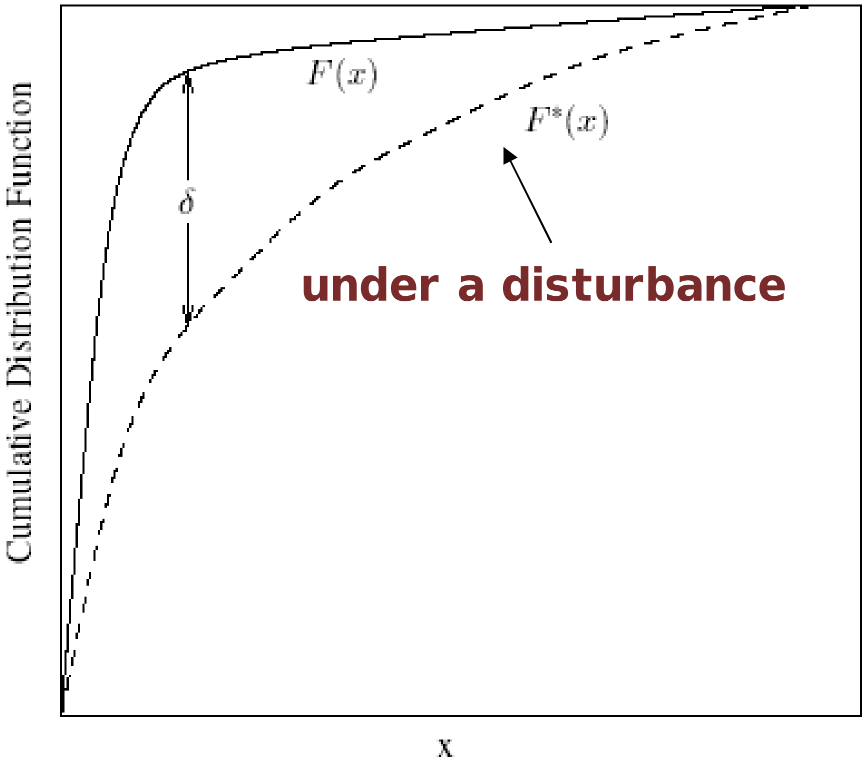
Policy	Shortest queue	Robust scheduling
Mean	18.3	9.1
Variance	1488	538
25%	0	0
50%	1.5	0
75%	18.8	5.3
95%	90.3	54.7
99%	191.9	108.8

– very simple policy produced dramatically better results

Measurement

- Great – this makes intuitive sense, but how do you measure it?

Robustness Metric

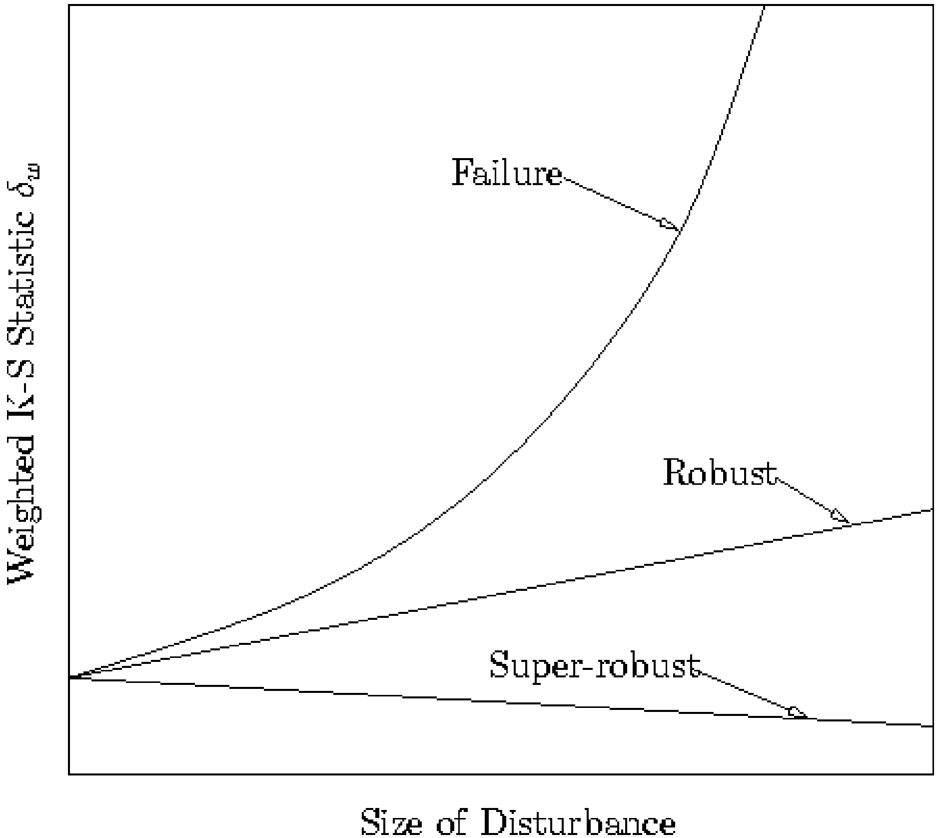


$$\delta = \sup_{-\infty < x < \infty} F(x) - F^*(x)$$

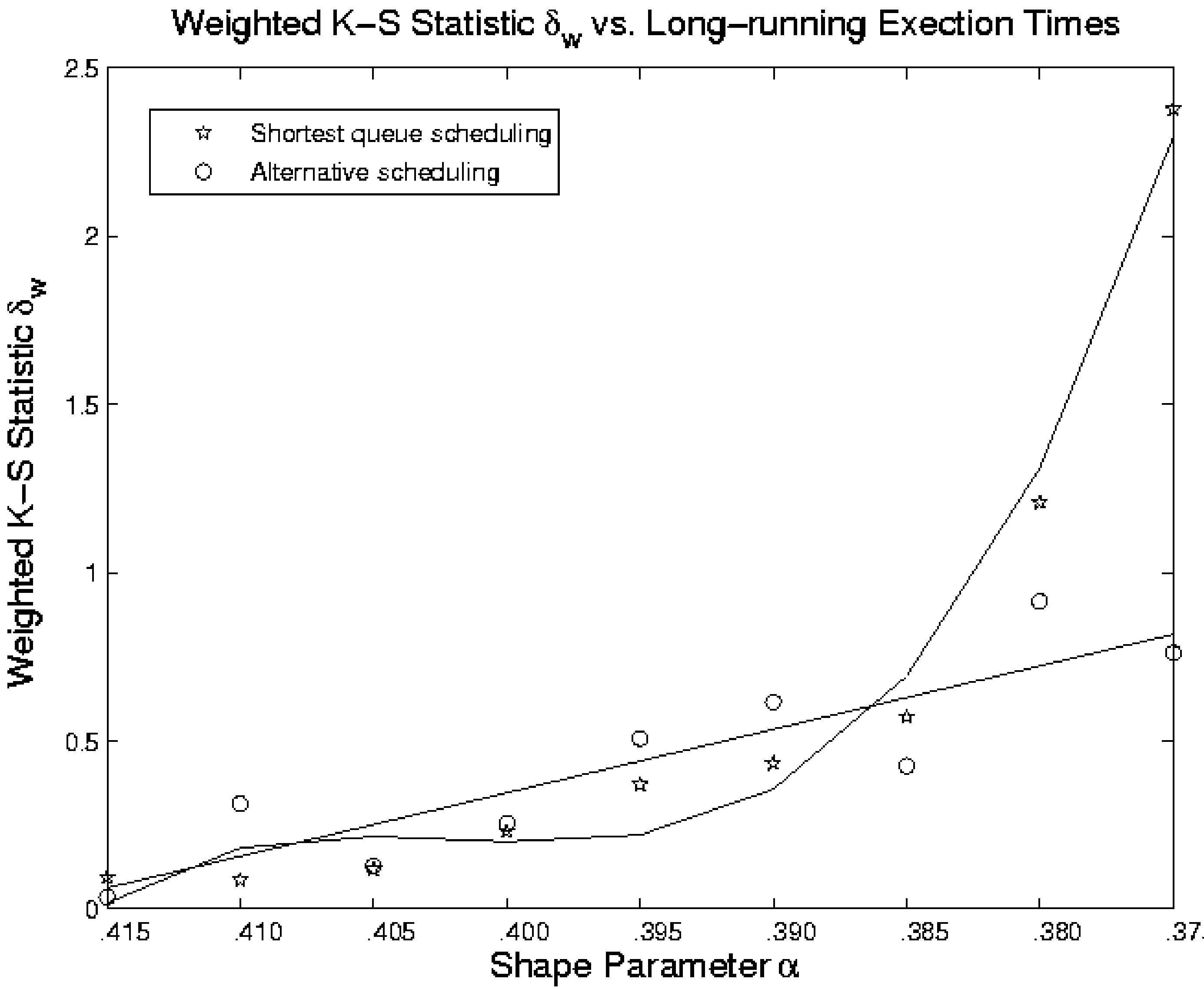
max

- Metric gives us a tool to analyze behavior
- Also useful for exploring system alternatives
  - scheduling algorithms
  - resource provisioning
  - etc

Metric (cont'd)

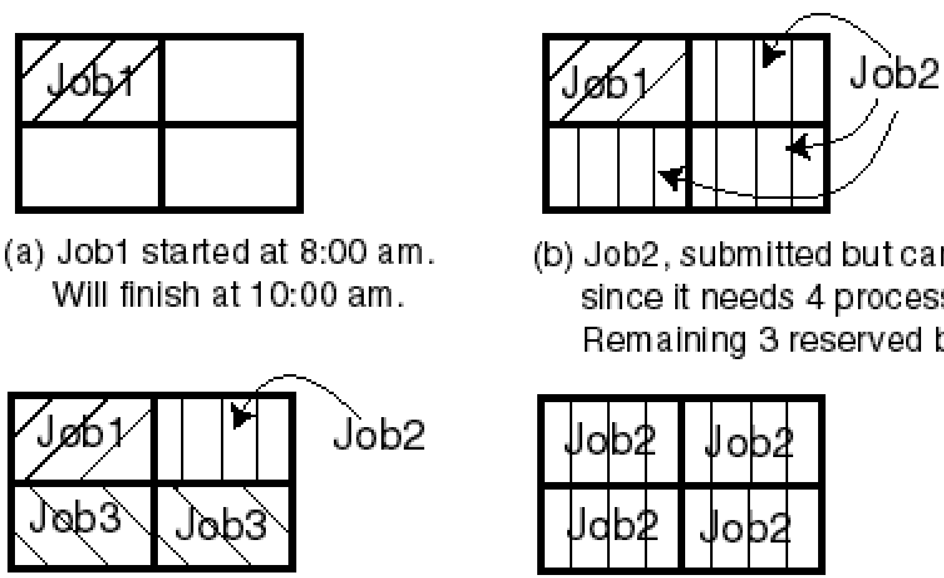


Robust to Heavy-Tailed Property



Backfiling

- Requires user estimates of parallel job run-time and # of desired processors
- Batch mode scheduling



(a) Job1 started at 8:00 am. Will finish at 10:00 am.

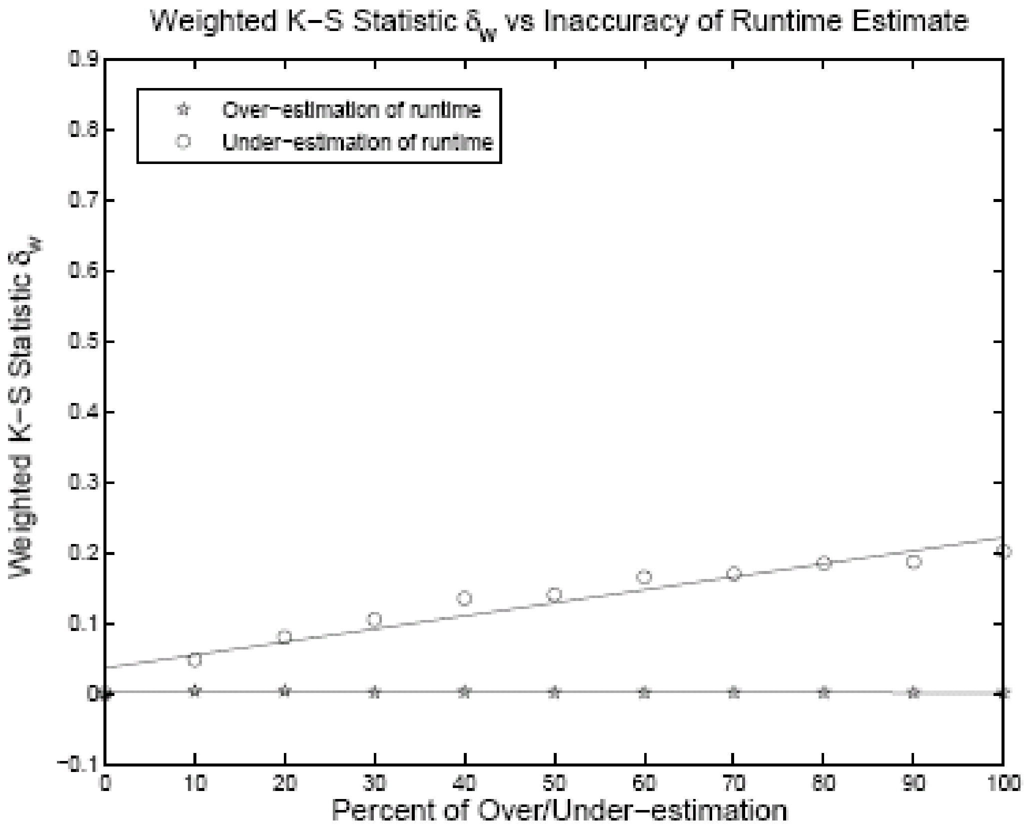
(b) Job2, submitted but can't start since it needs 4 processors. Remaining 3 reserved by Job2.

(c) At 8:30 am Job3 submitted. Job3 backfills Job2.

(d) At 10:00 am, Job2 starts.

Backfiling (cont'd)

Disturbance: user under- or over-estimation in waiting time (user behavior
SDSC supercomputer job trace



Robustness as a function of <u>waiting time</u>

Video Server

Disturbance: demand for videos; performance metric is response time

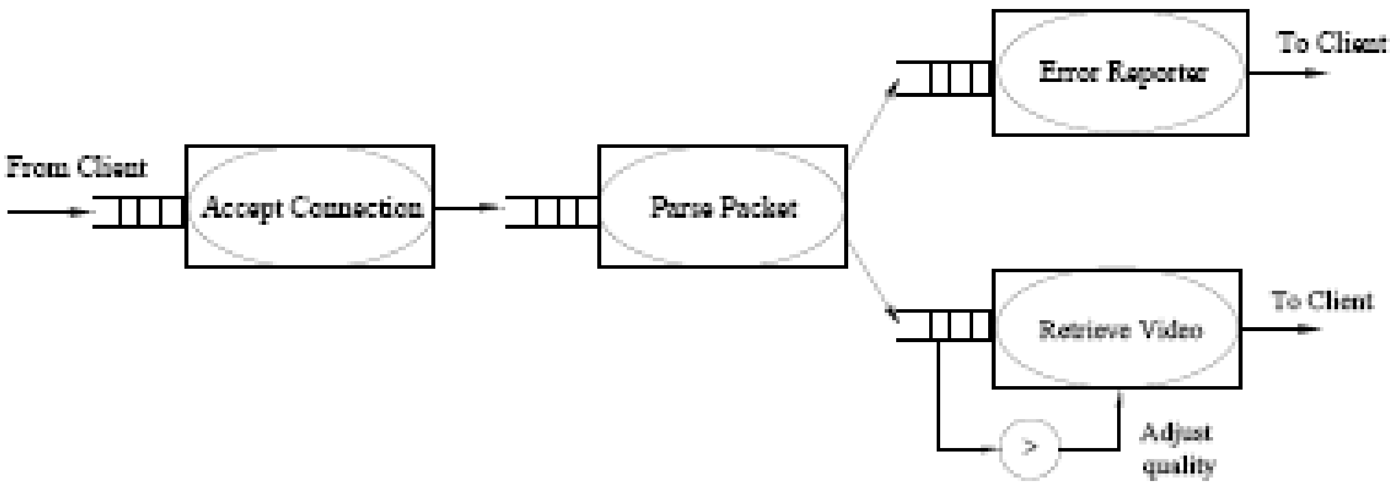
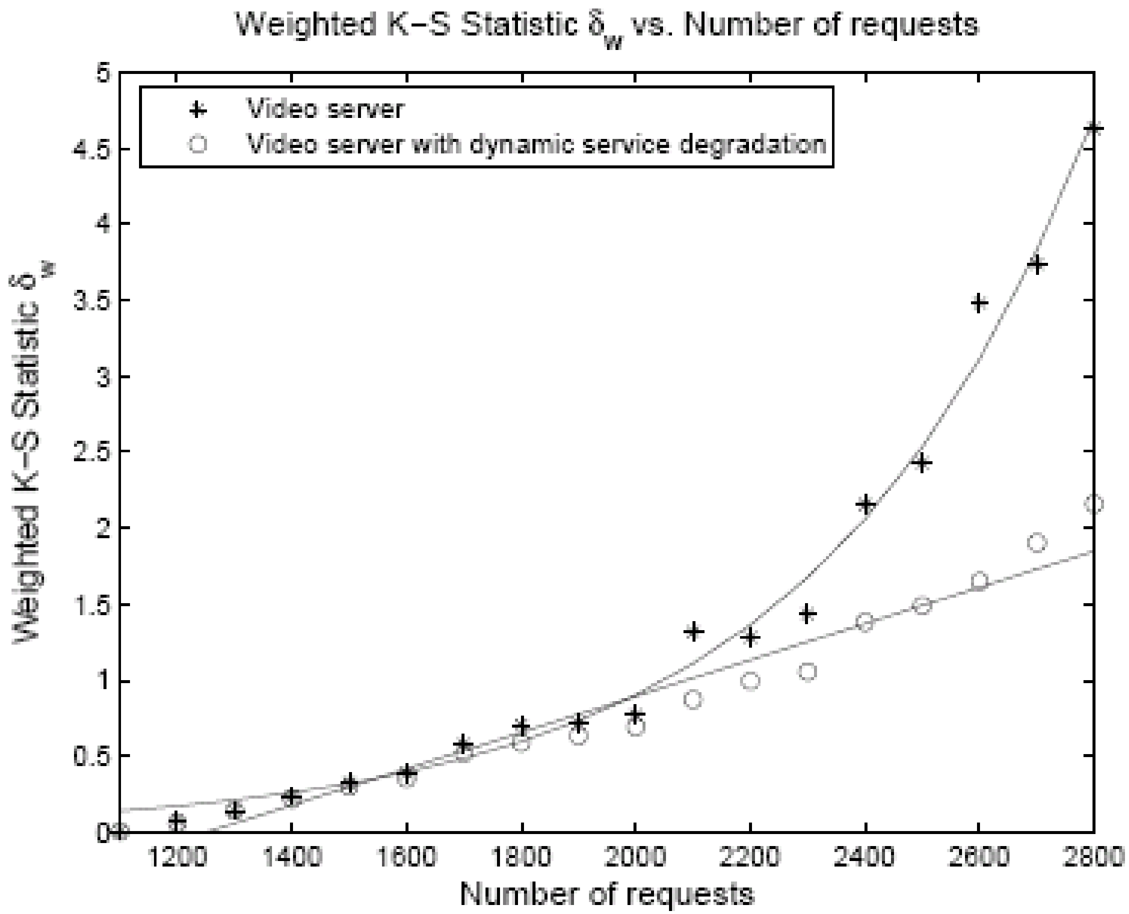


Figure 6. SEDA-based Video Server

Video Server (cont'd)



Robustness as a function of <u>response time</u>

Cool Example: Sensor Nets

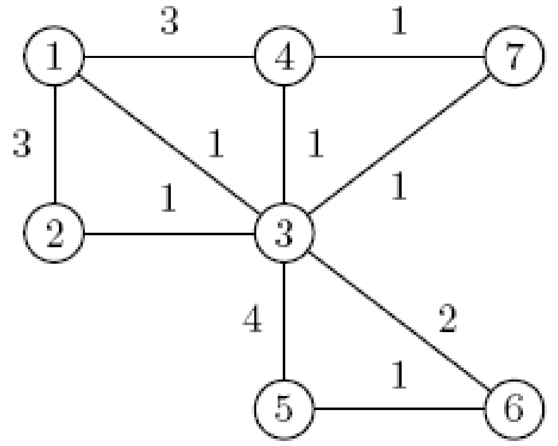


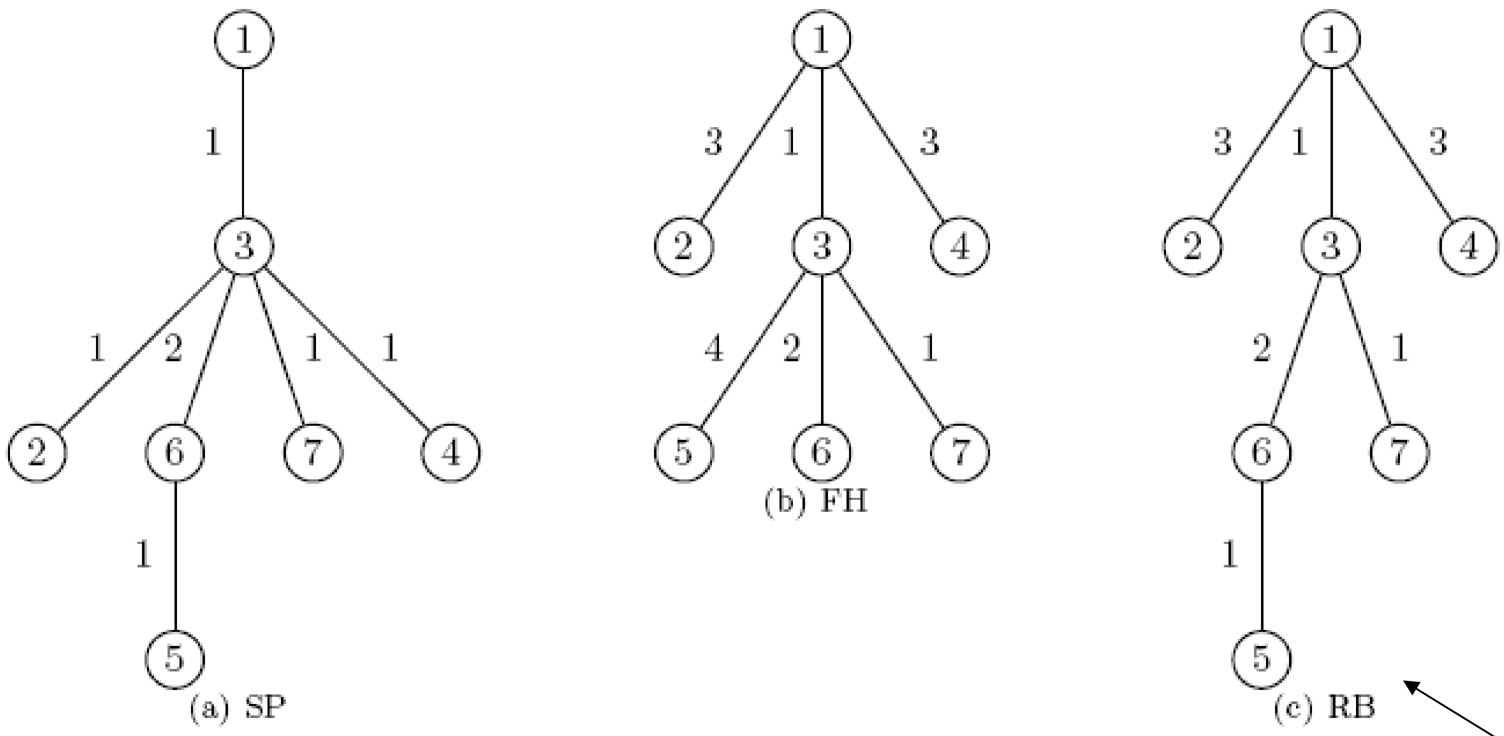
Figure 4.3: A Small Sensor Network

Goal is to route/aggregate all information to (1)
Links are weighted in terms of cost

Could also refer to any distributed application requiring data collection

Disturbance: node failure and data loss

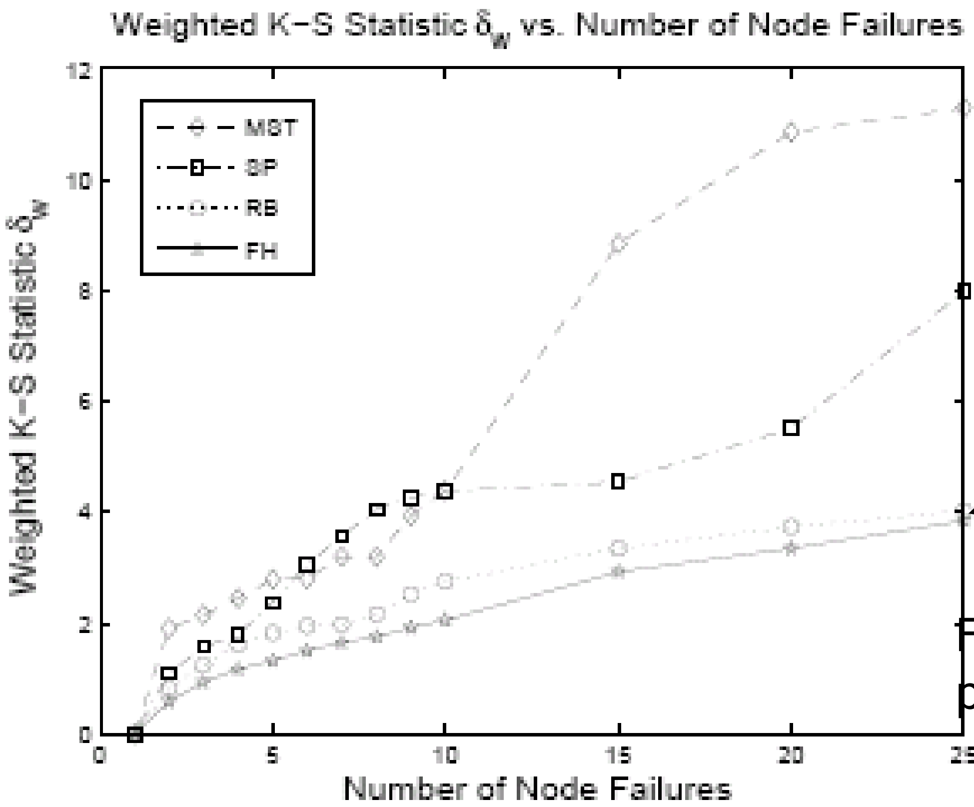
Sensor Net (cont'd)



Different spanning tree routing options

our robust approach

Results



ours

FH is also, but is
poor for battery power

Robustness as a function of <u>data survival</u>

Questions?

Dynamic Architectures

- Our distributed system architecture must be dynamic to cope with all the other forms of dynamism…
- Decentralization is the key

Background

- Grids are distributed … but also centralized
  - Condor, Globus, BOINC, Grid Services, VOs
  - Why? client-**server** based
- Centralization pros
  - Security, policy, global resource management
- Decentralization pros
  - **Reliability**, dynamic, flexible, **scalable**

Challenges

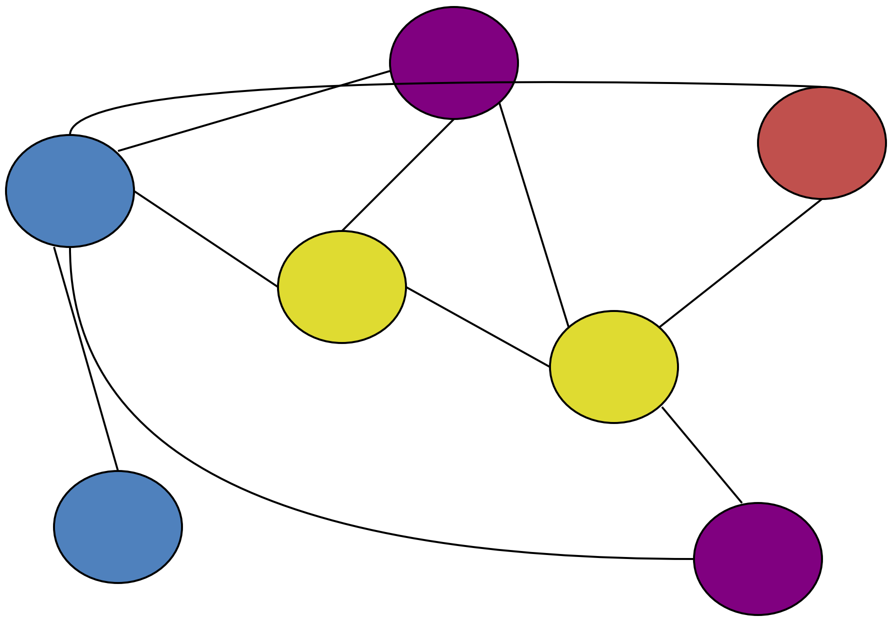
- May have to live within the Grid ecosystem
  - Condor, Globus, Grid services, VOs, etc.
  - First principle approaches are risky (Legion)

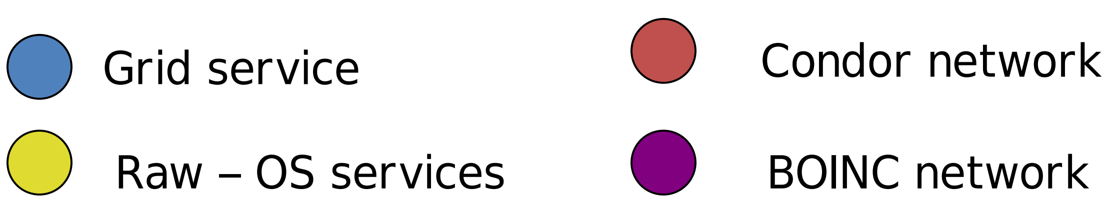
- 50K foot view
  - How to decentralize Grids yet retain their existing features?
  - High performance, workflows, performance prediction, etc.

Decentralized Grid platform

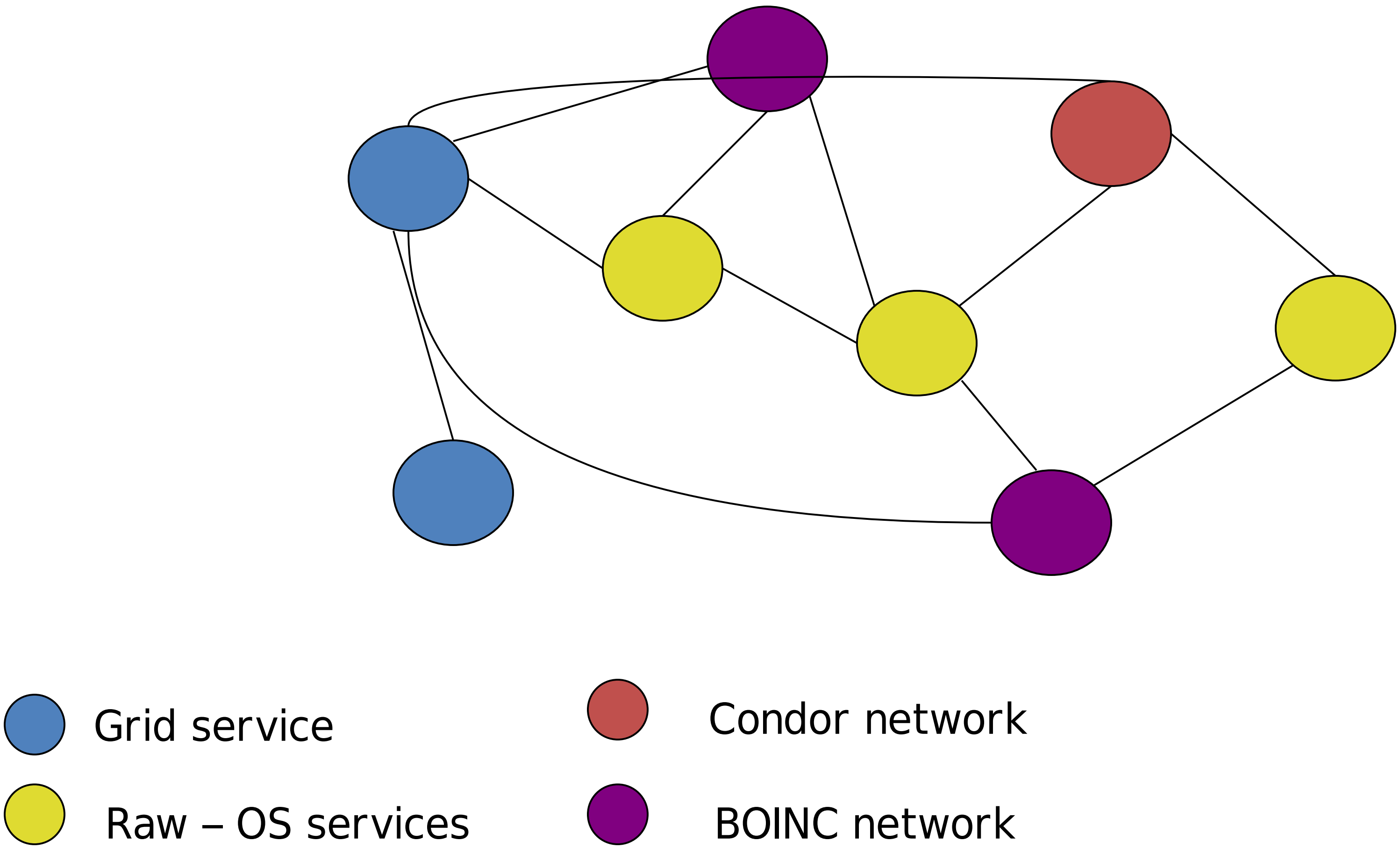
- Minimal assumptions about each "node"
- Nodes have associated "assets" (A)
  - basic: CPU, memory, disk, etc.
  - complex: application services
  - exposed interface to assets: OS, Condor, BOINC, Web service
- Nodes may up or down
- Node trust is not a given (do X, does Y instead)
- Nodes may connect to other nodes or not
- Nodes may be aggregates
- Grid may be large > 100K nodes, scalability is key

Grid Overlay

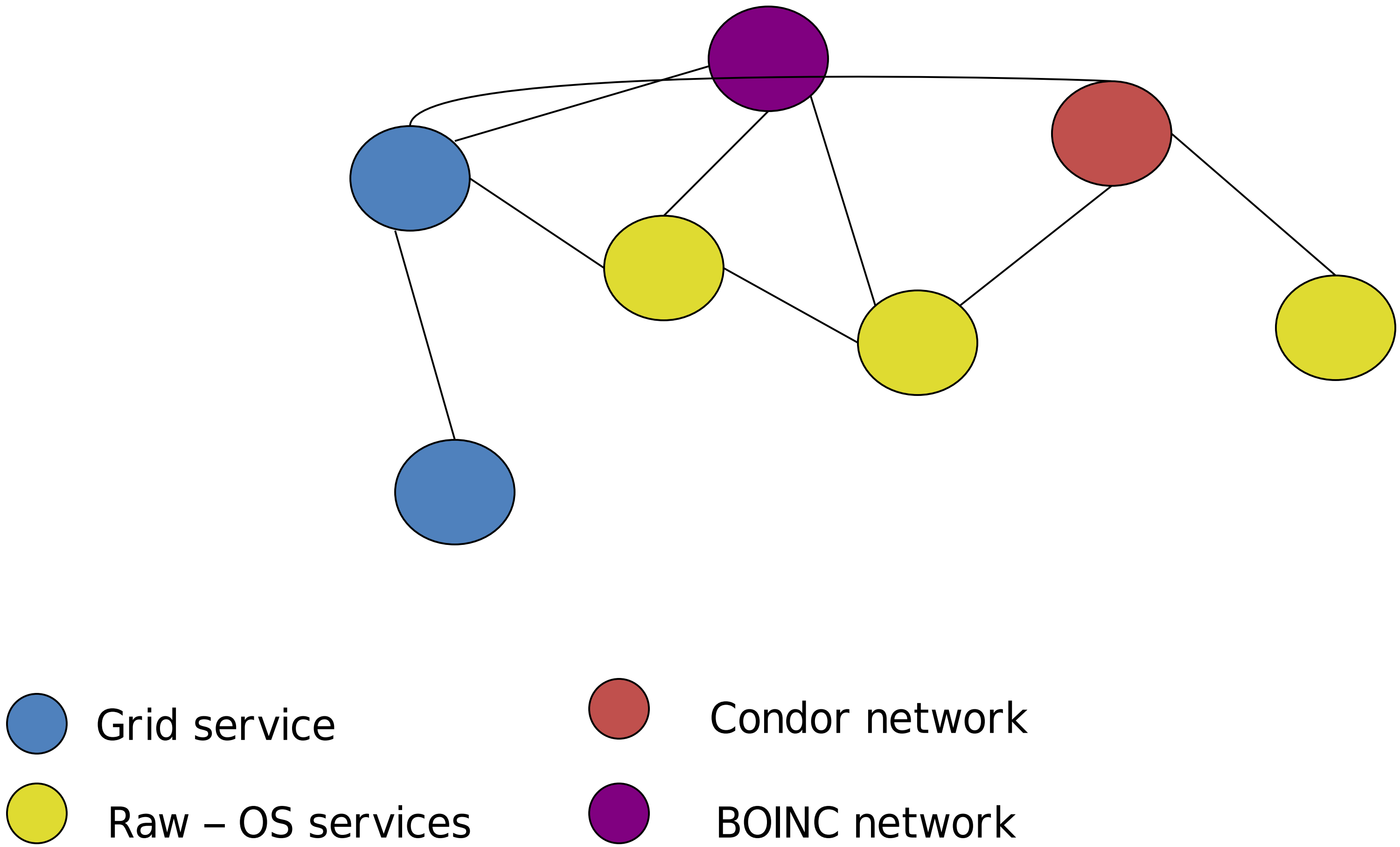




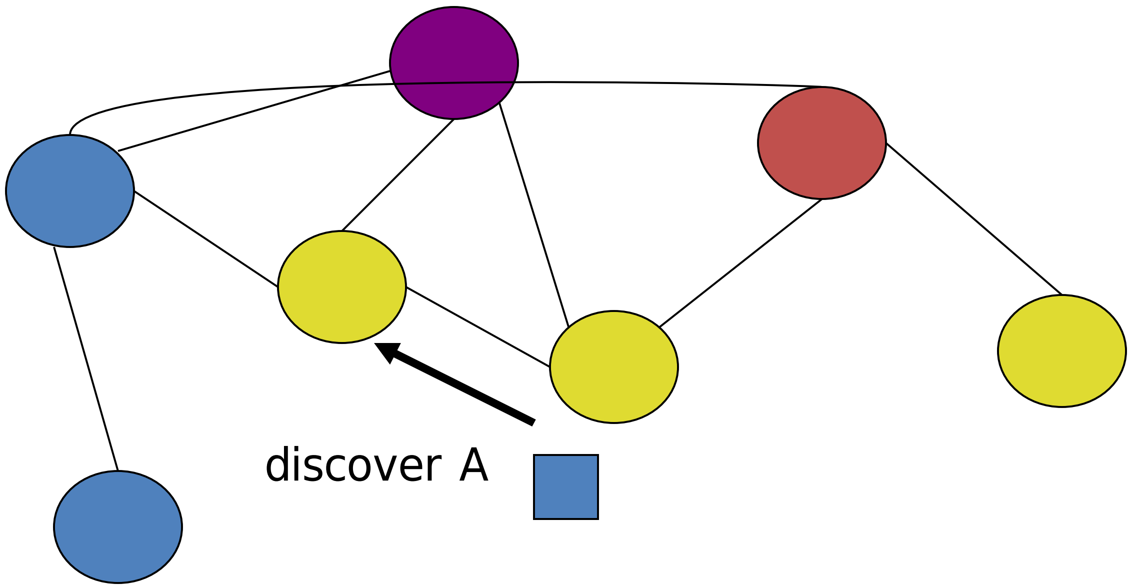
Grid Overlay - Join



Grid Overlay - Departure

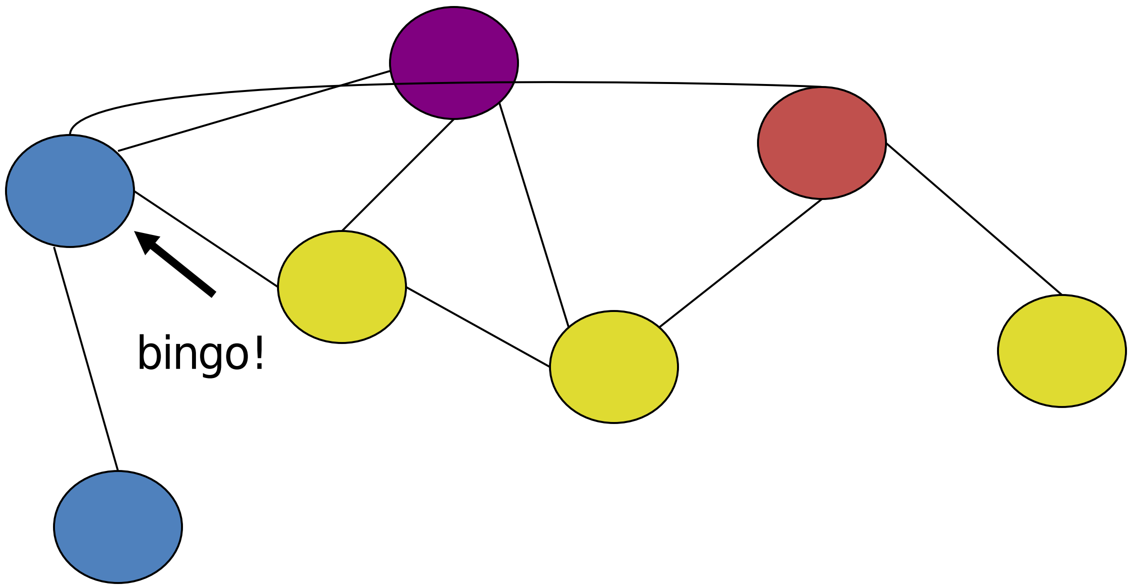


Routing = Discovery

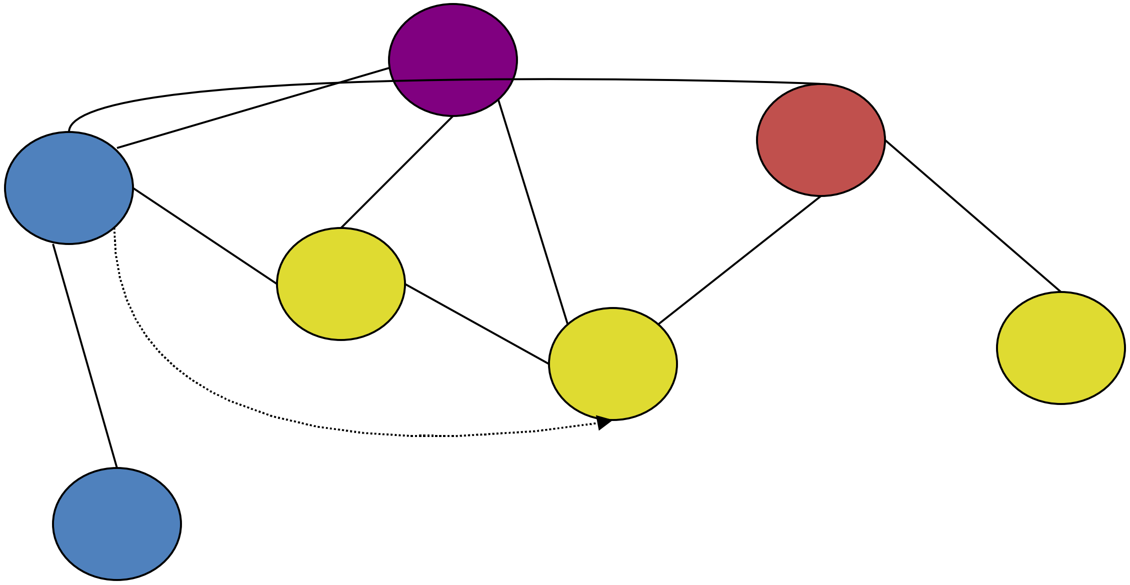


Query contains sufficient information to locate a node: RSL, ClassAd, etc
Exact match or semantic match

Routing = Discovery



Routing = Discovery



Discovered node returns a handle sufficient for the “client” to interact with it

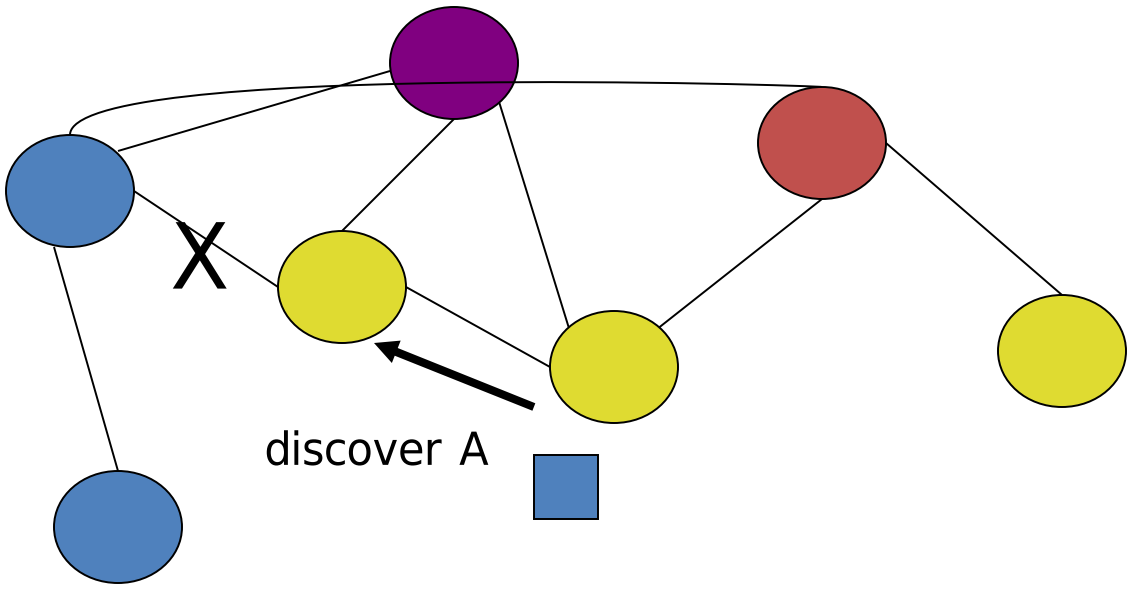
- perform service invocation, job/data transmission, etc

Routing = Discovery

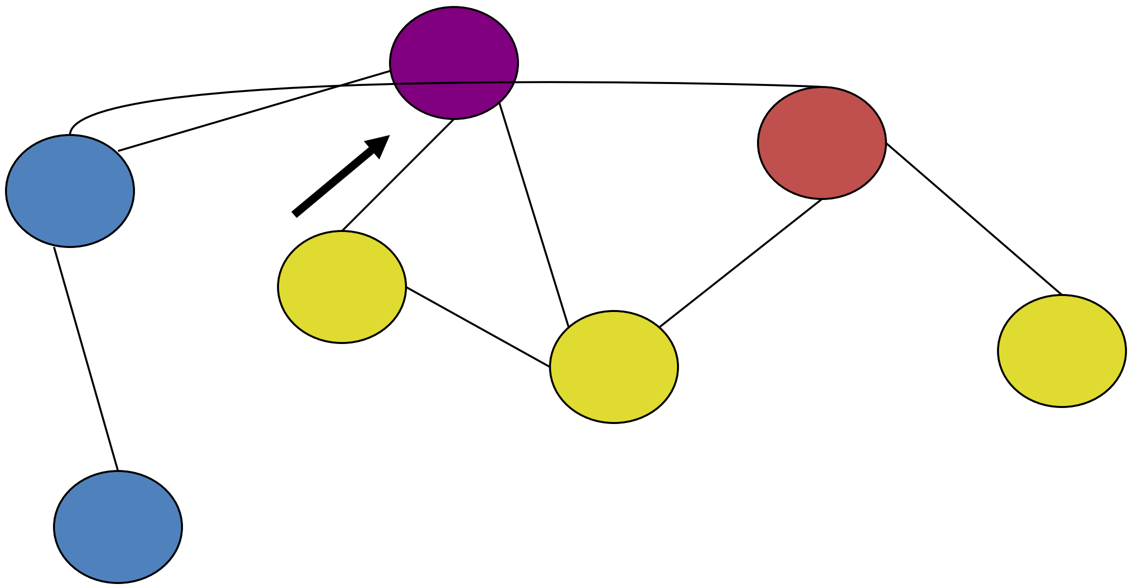
- Three parties
  - *initiator* of discovery events for A
  - *client*: invocation, health of A
  - *node* offering A
- Often initiator and client will be the same

- Other times client will be determined dynamically
  - if W is a web service and results are returned to a calling client, want to locate C_W near W =>
  - discover W, then C_W !

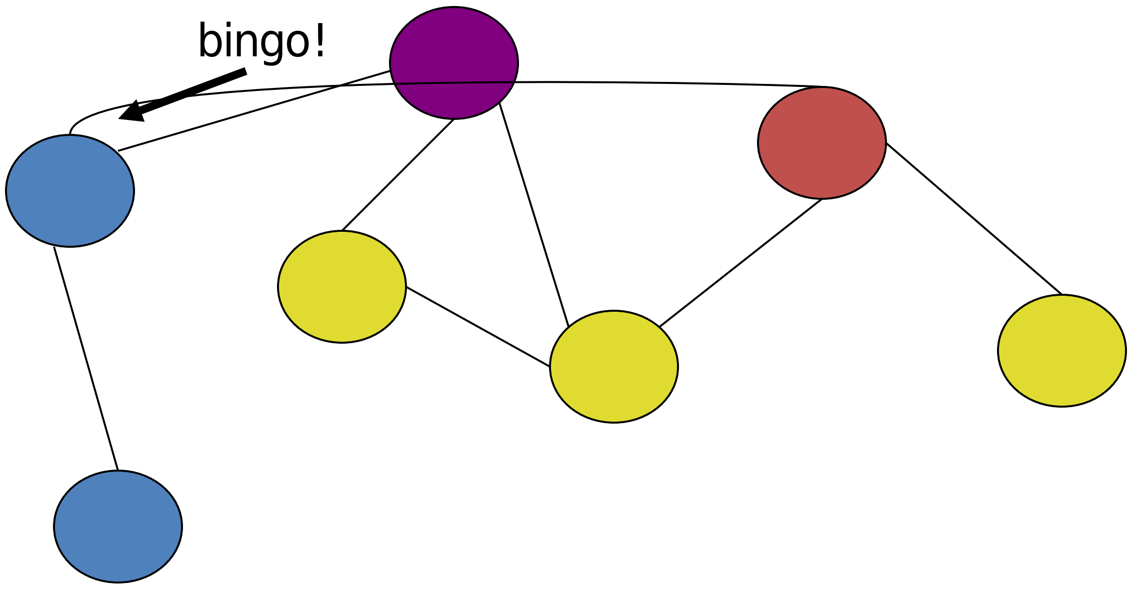
Routing = Discovery



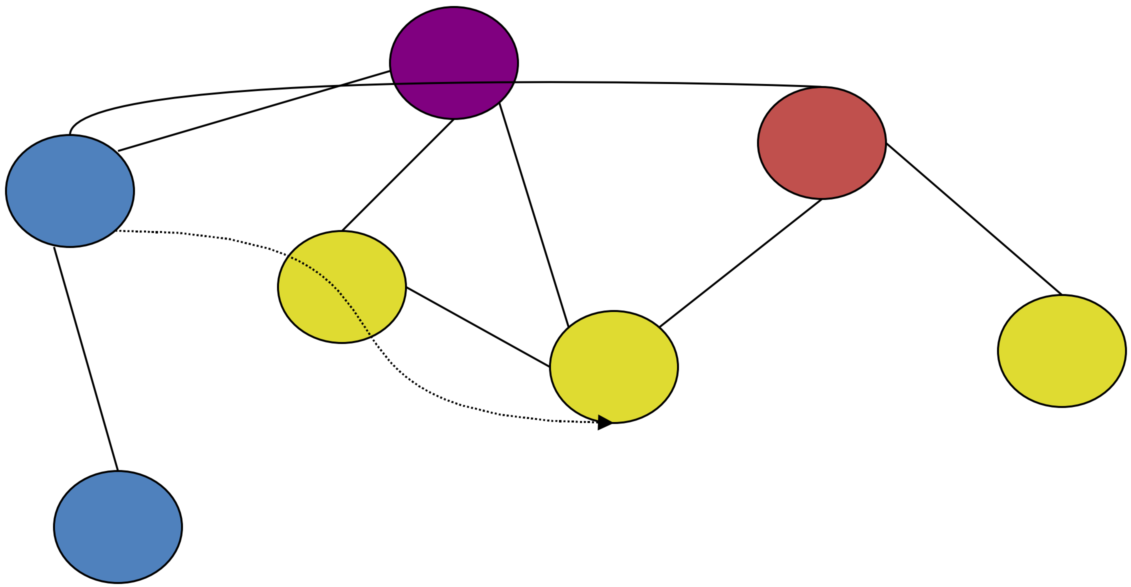
Routing = Discovery



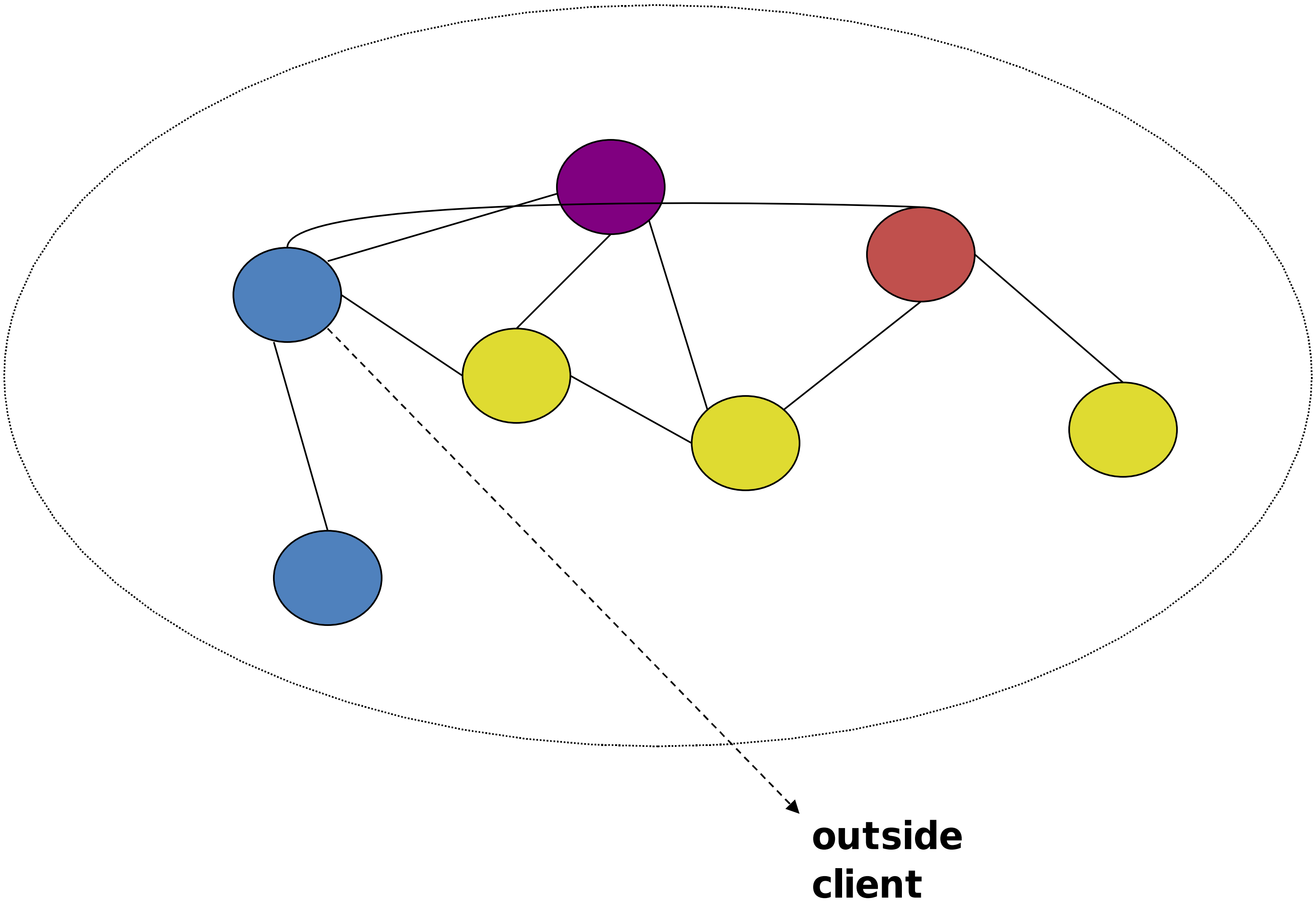
Routing = Discovery



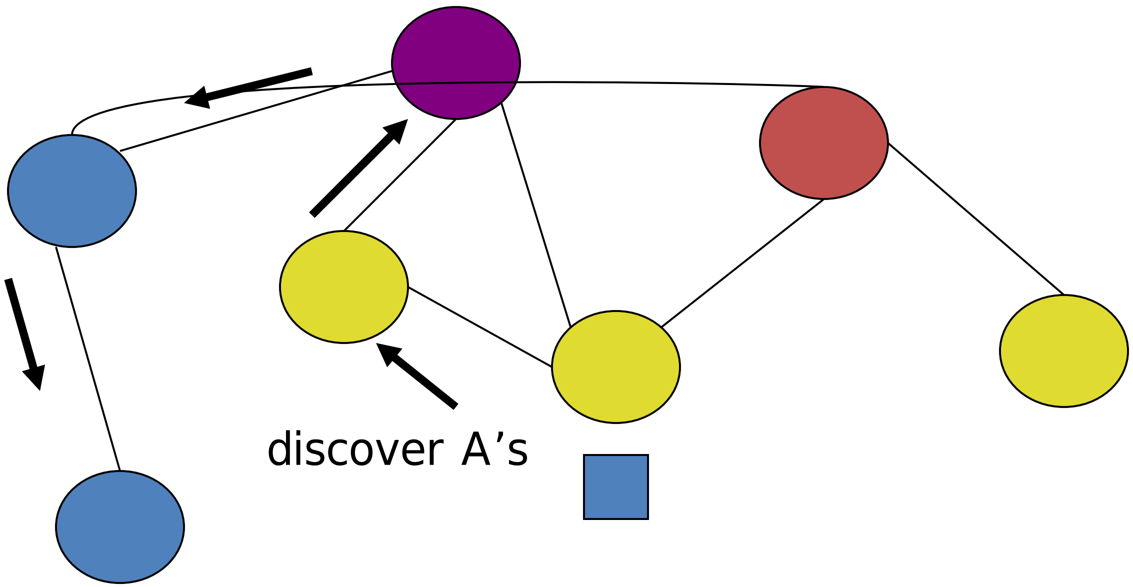
Routing = Discovery



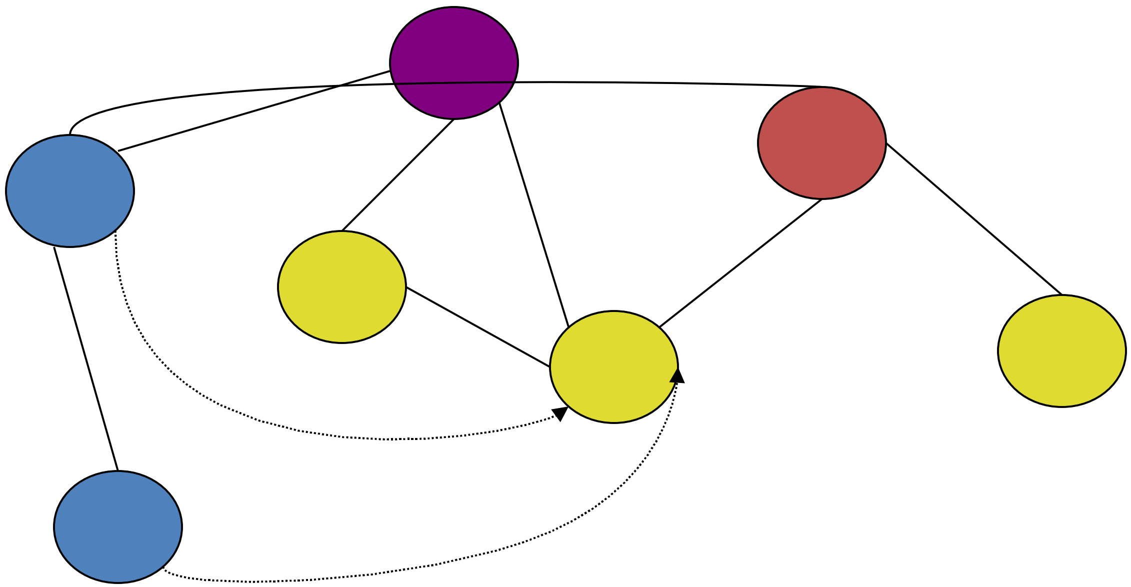
Routing = Discovery



Routing = Discovery



Routing = Discovery



Grid Overlay

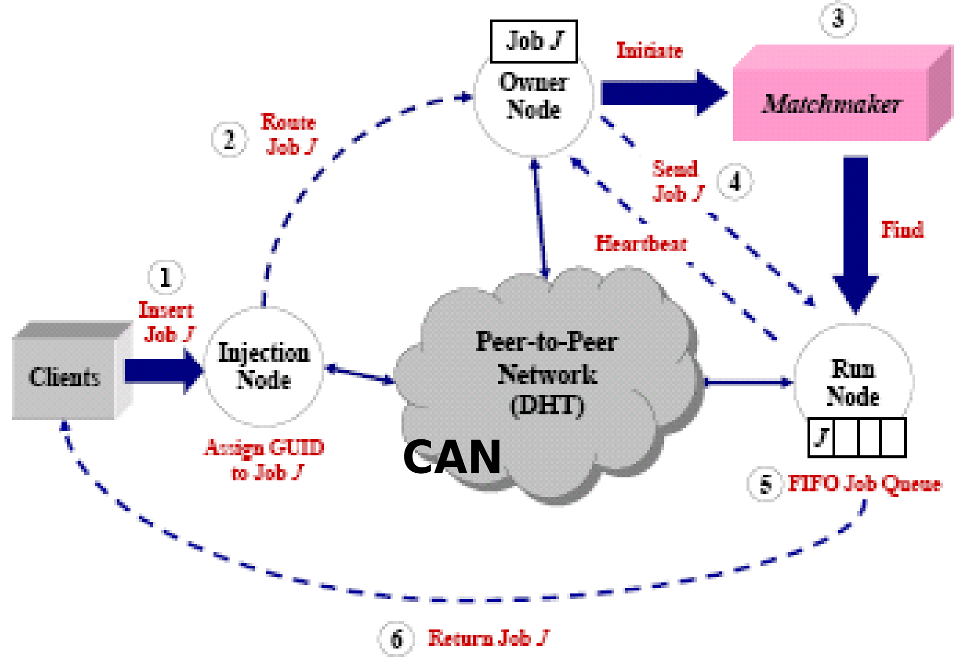
- This generalizes …
  - Resource query (query contains job requirements)
  - Looks like decentralized "matchmaking"
- These are the easy cases …
  - independent simple queries
    - find a CPU with characteristics x, y, z
    - find 100 CPUs each with x, y, z
  - suppose queries are complex or related?
    - find N CPUs with aggregate power = G Gflops
    - locate an asset near a prior discovered asset

Grid Scenarios

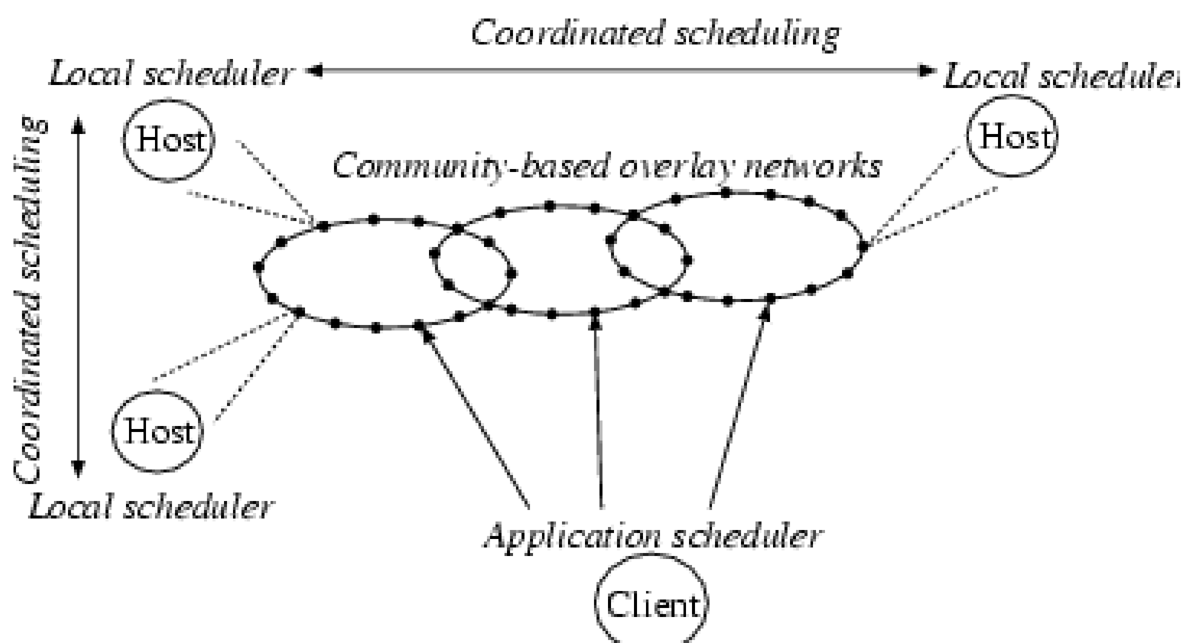
- Grid applications are more challenging
  - Application has a more complex structure - multi-task, parallel/distributed, **control/data** dependencies
    - individual job/task needs a resource **near** a data source
    - workflow
    - queries are not independent
  - Metrics are collective
    - not simply raw throughput
    - makespan
    - response
    - QoS

Related Work

- Maryland/Purdue
  - matchmaking



- Oregon-CCOF
  - time-zone



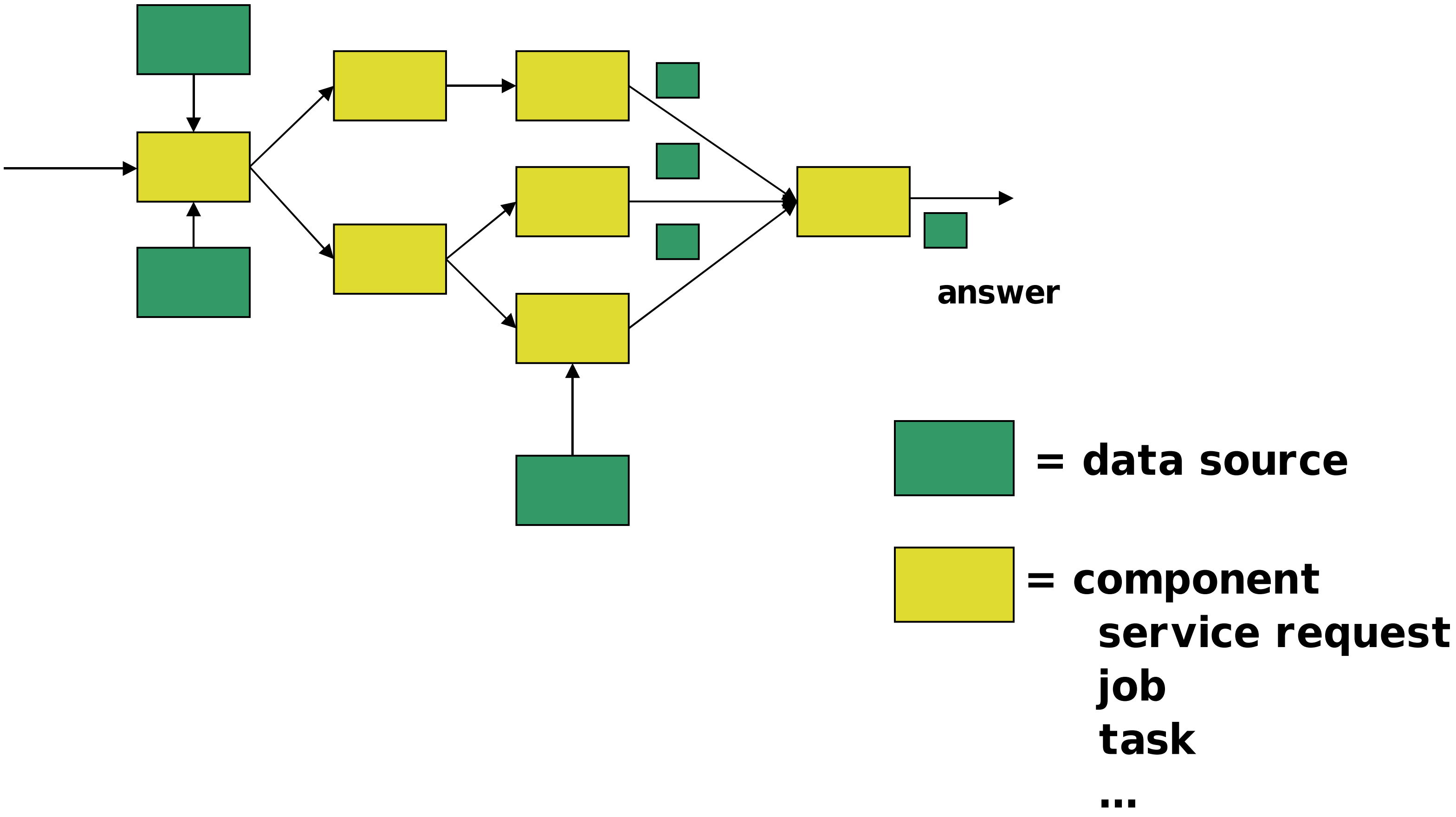
Related Work (cont'd)

- None of these approaches address the Grid scenarios (in a decentralized manner)
  - Complex multi-task data/control dependencies
  - Collective metrics

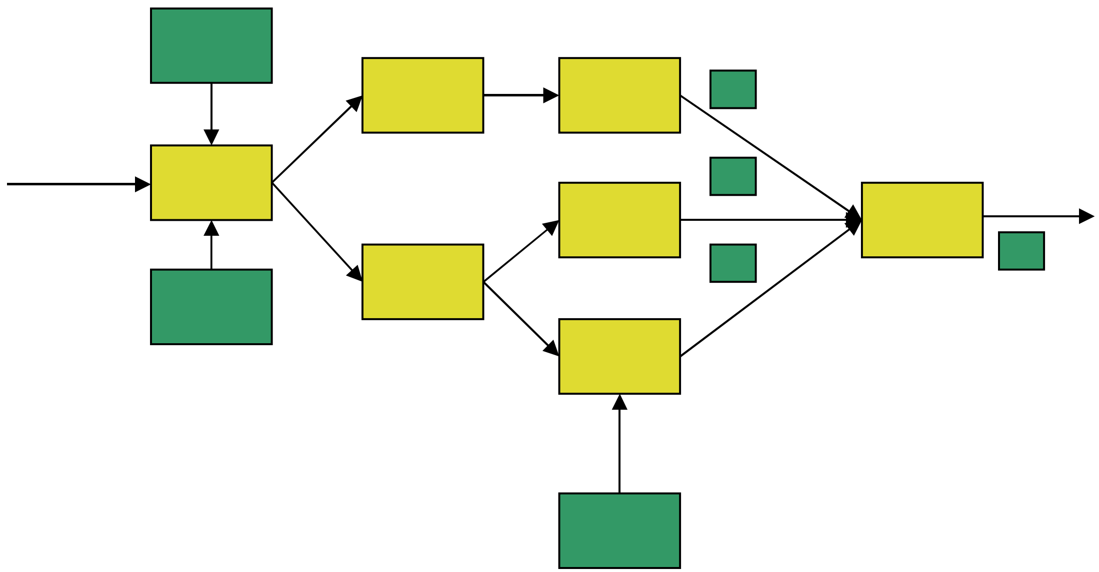
50K Ft Research Issues

- **Overlay Architecture**
  - structured, unstructured, hybrid
  - what is the right architecture?
- **Decentralized control/data dependencies**
  - how to do it?
- **Reliability**
  - how to achieve it?

Context: Application Model

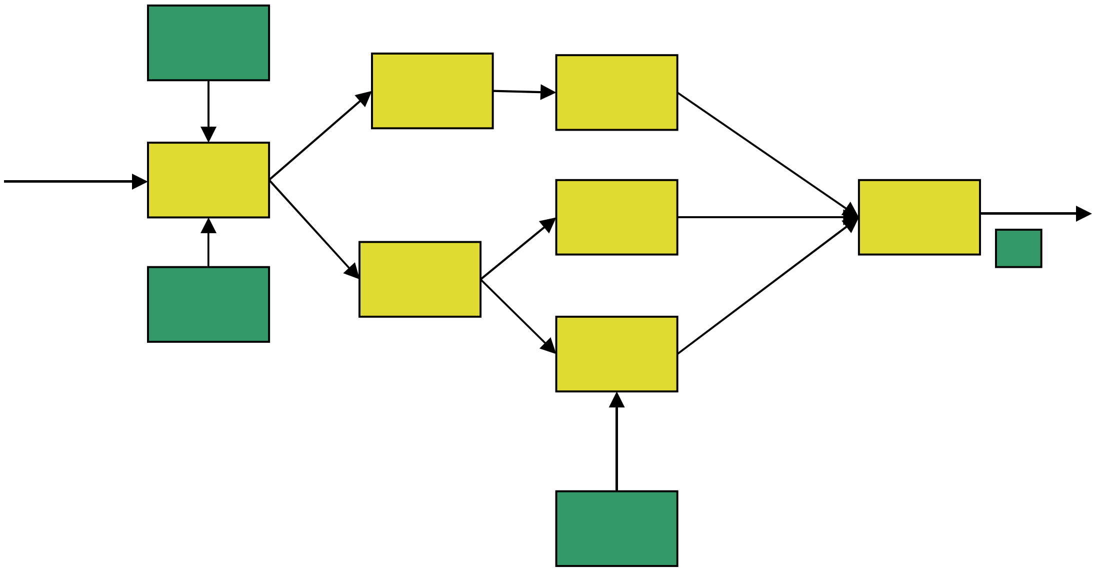


Context: Application Models



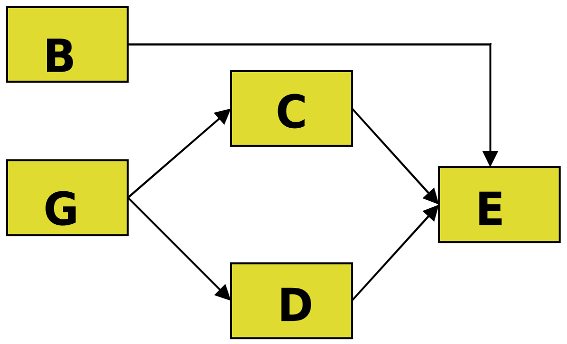
Reliability
Control dependence
Collective metrics

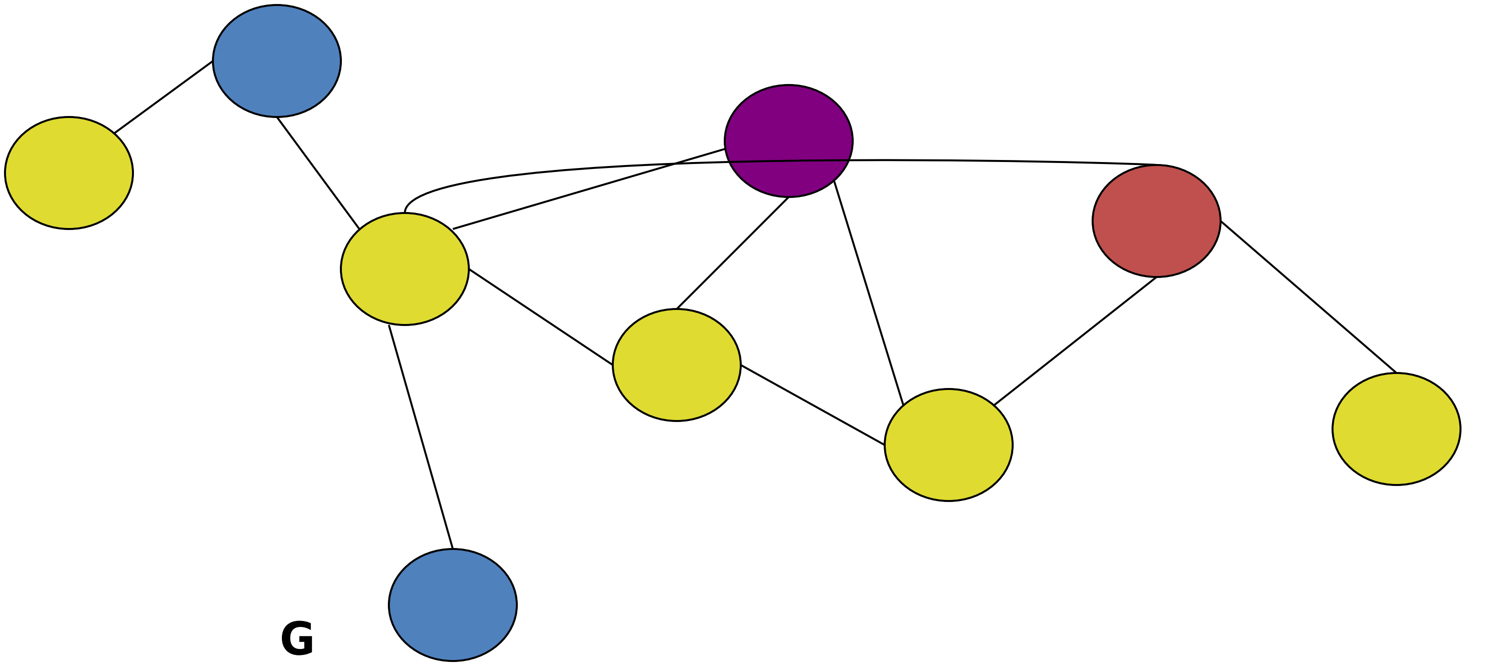
Application Models



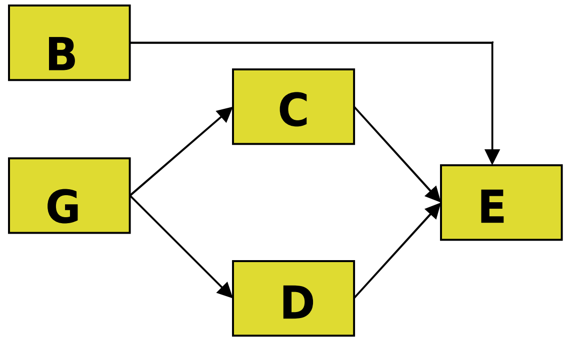
Reliability
Control dependence
Collective metrics

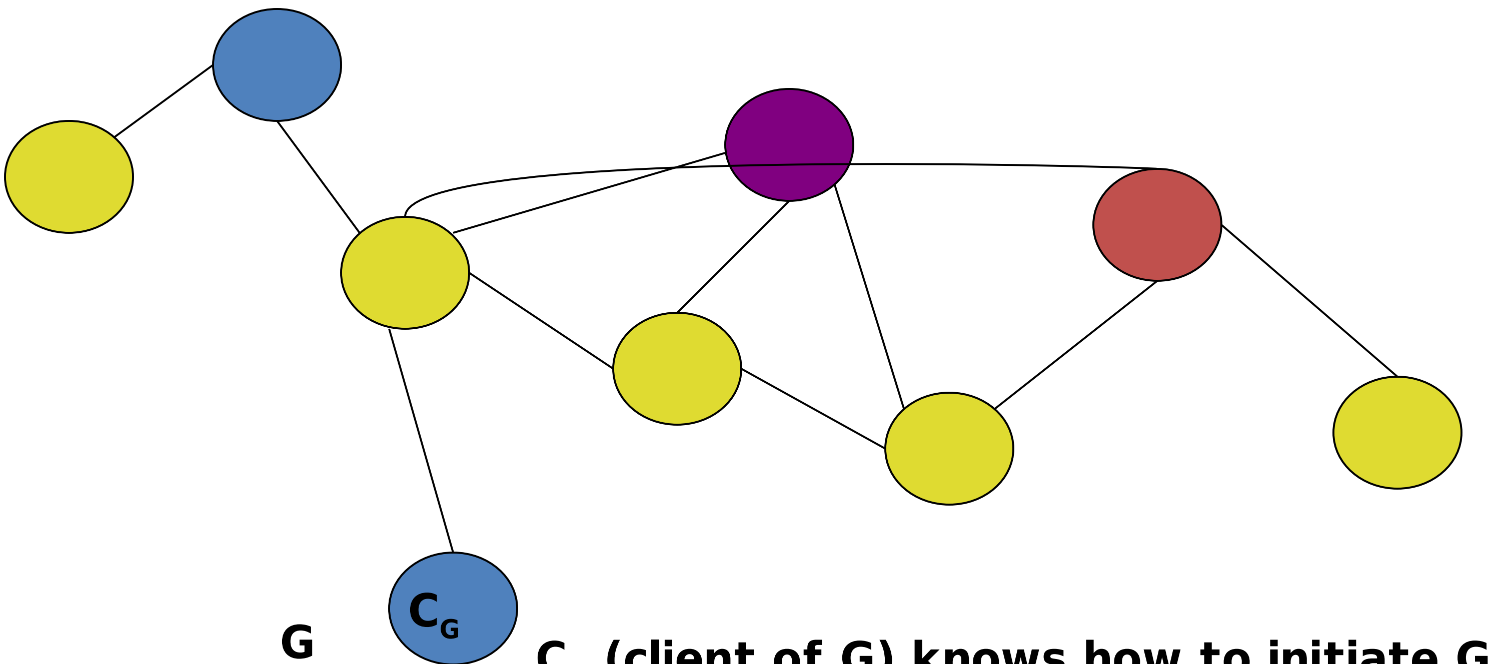
Reliability Example





Reliability Example

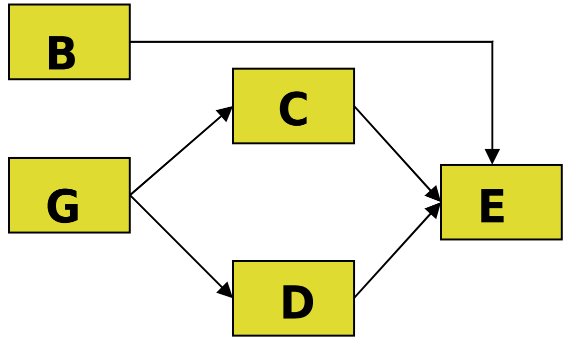


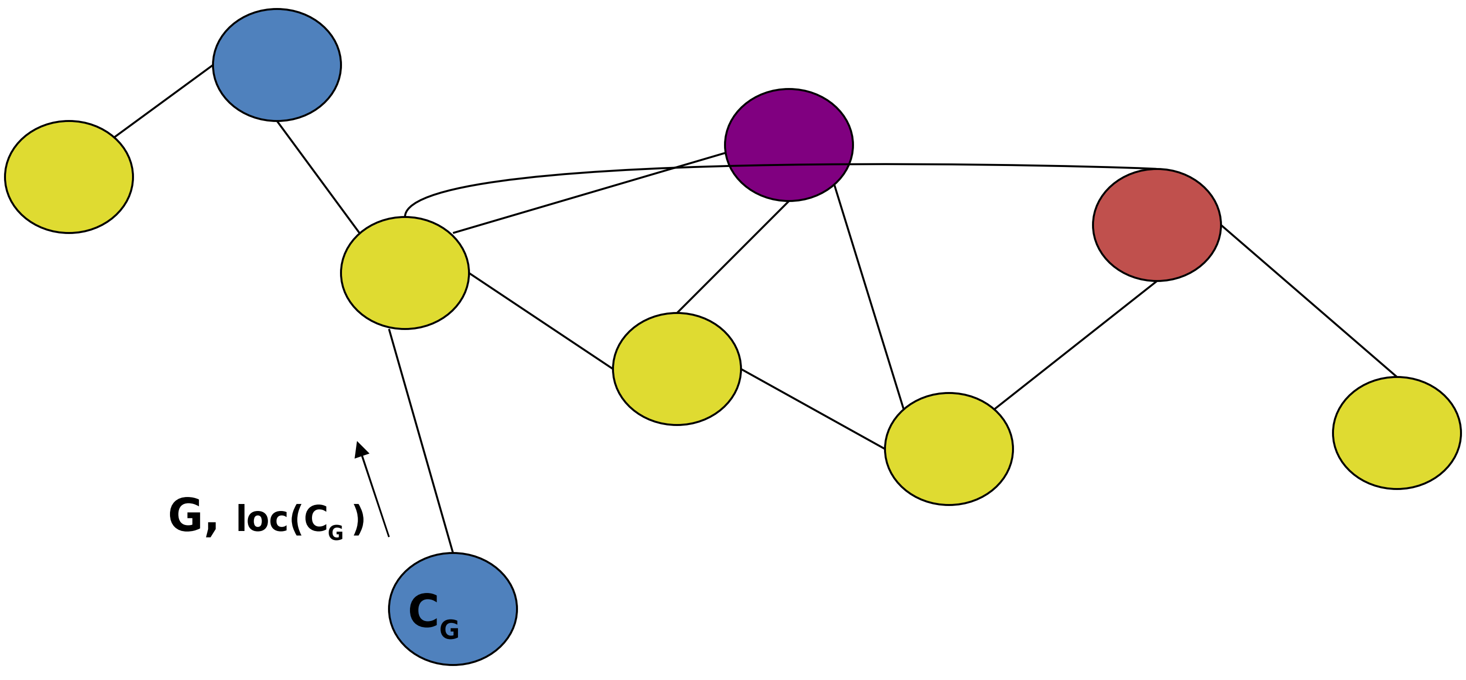


C_G (client of G) knows how to initiate G

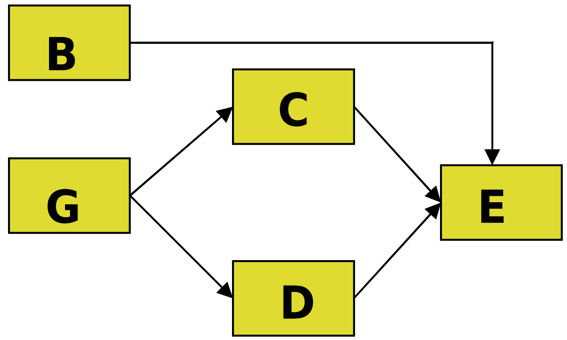
C_G also responsible for G's health

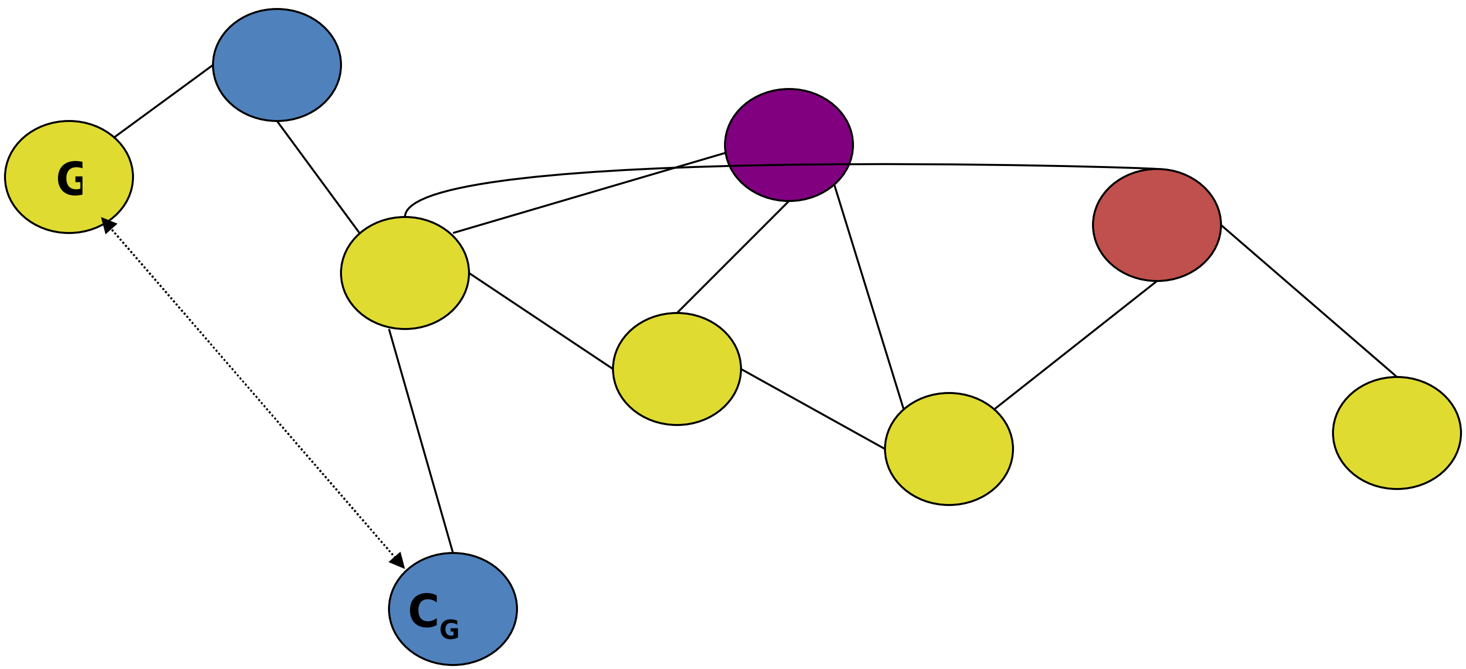
Reliability Example



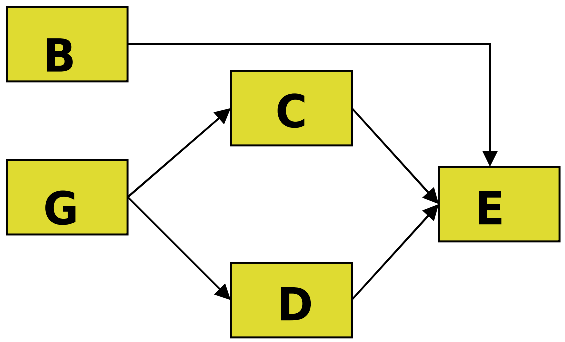


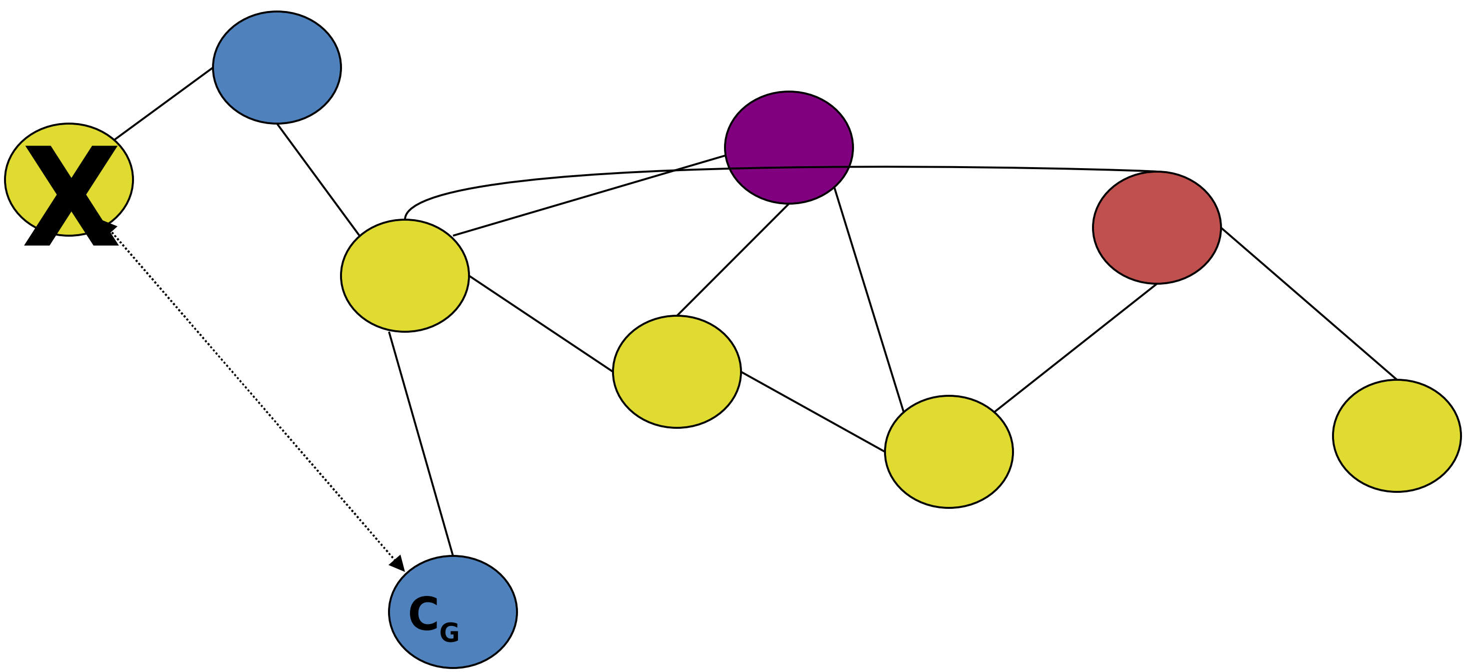
Reliability Example



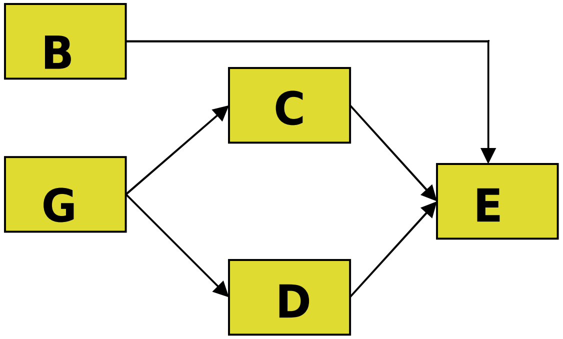


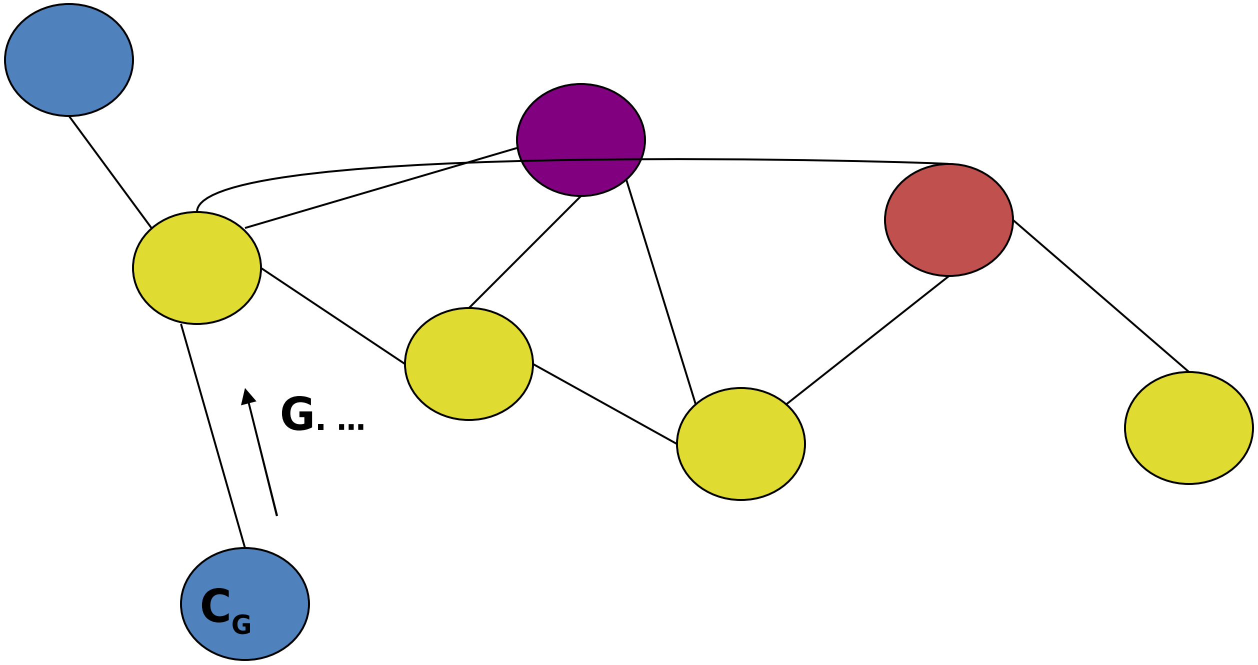
Reliability Example



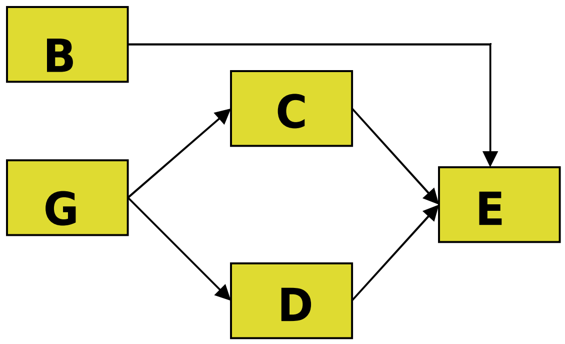


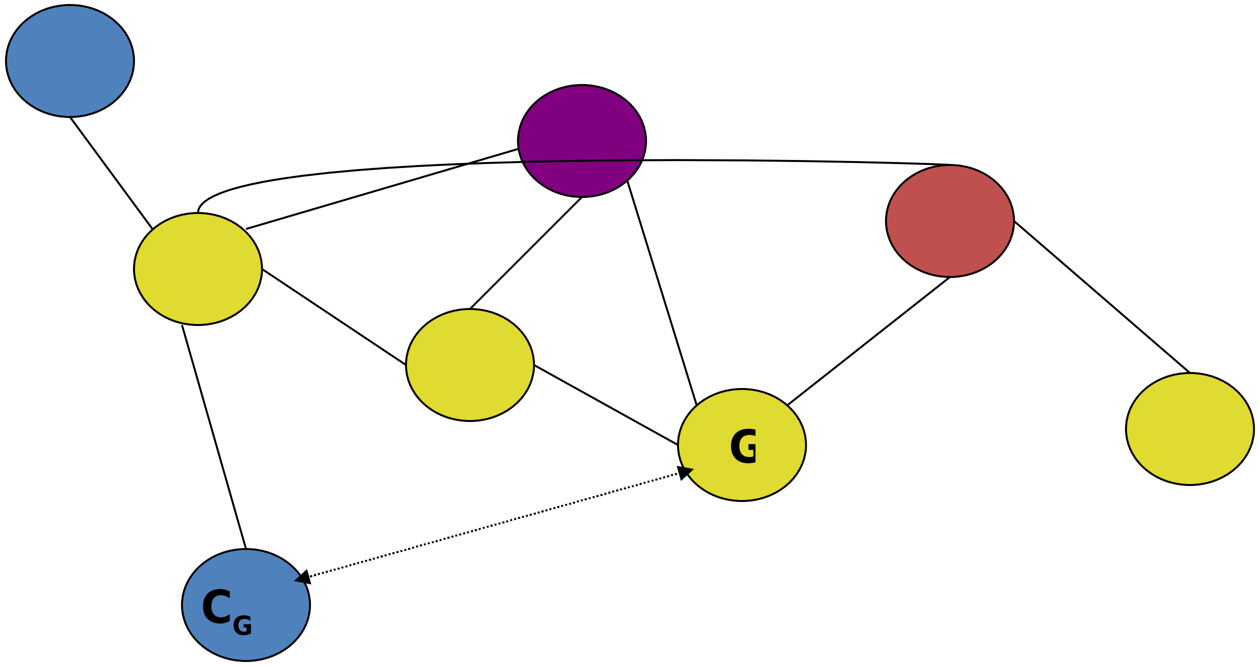
Reliability Example



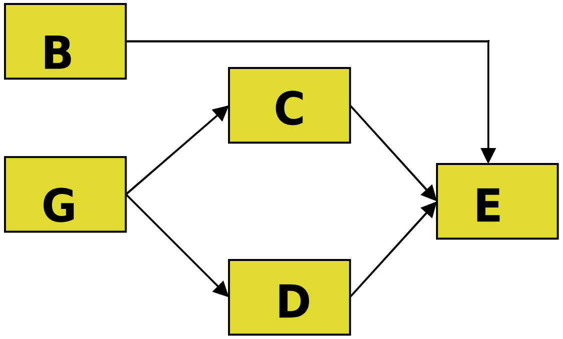


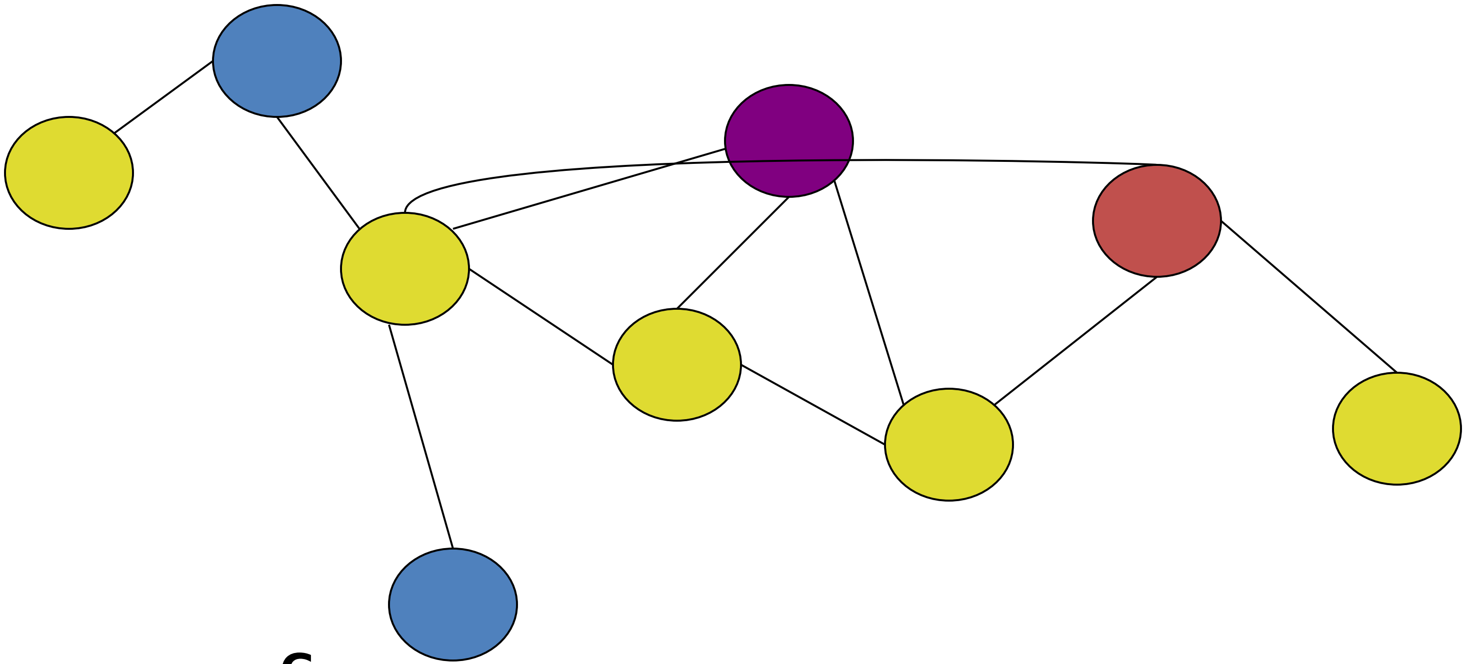
Reliability Example





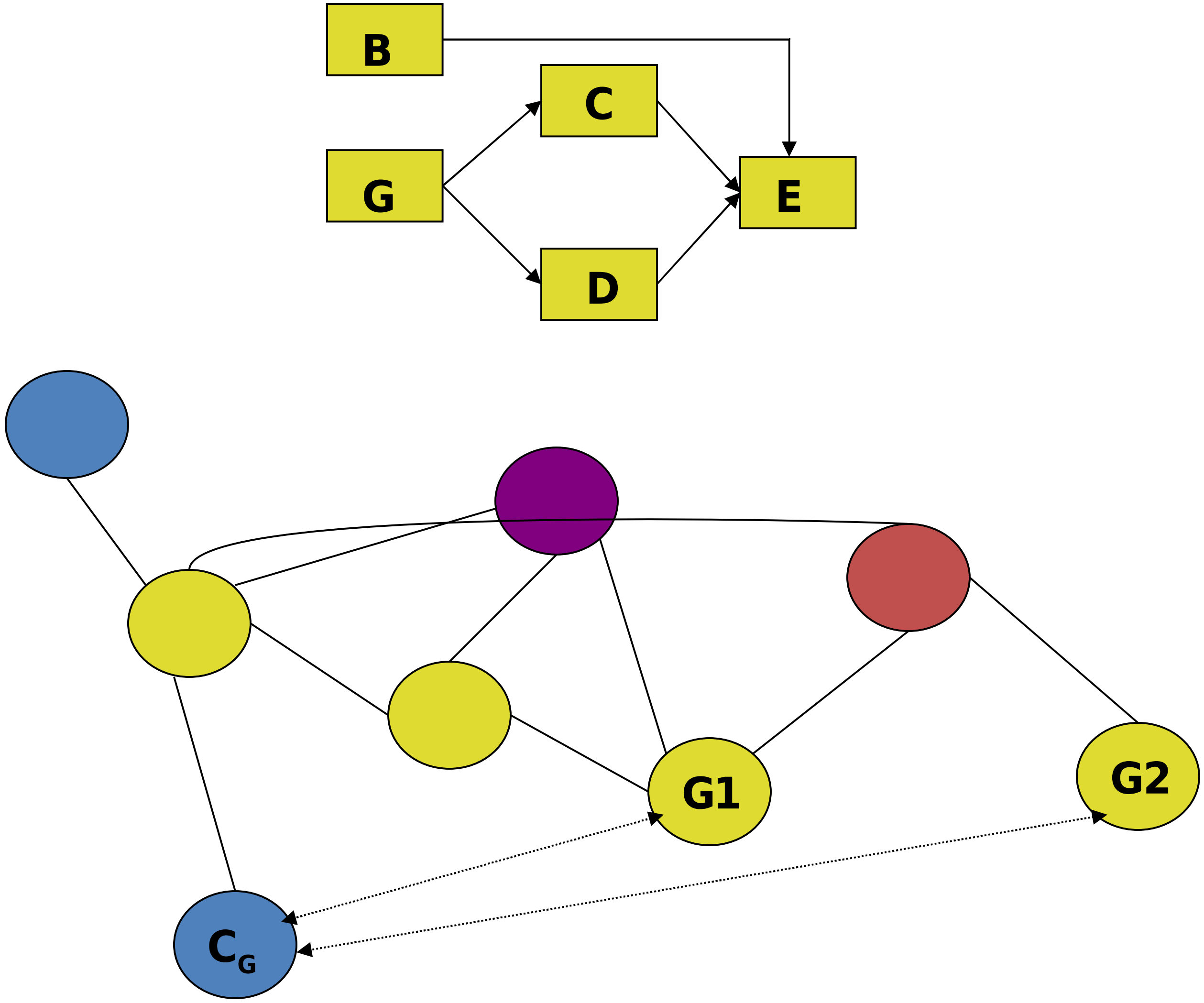
Component Replication





G

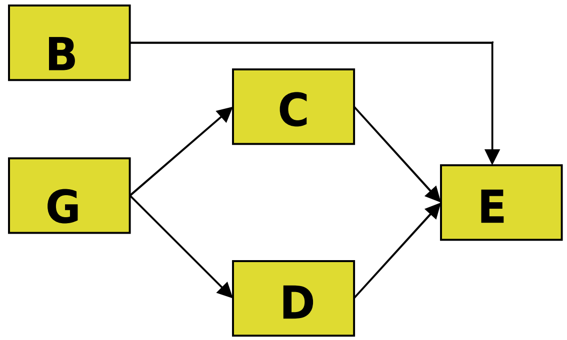
Component Replication

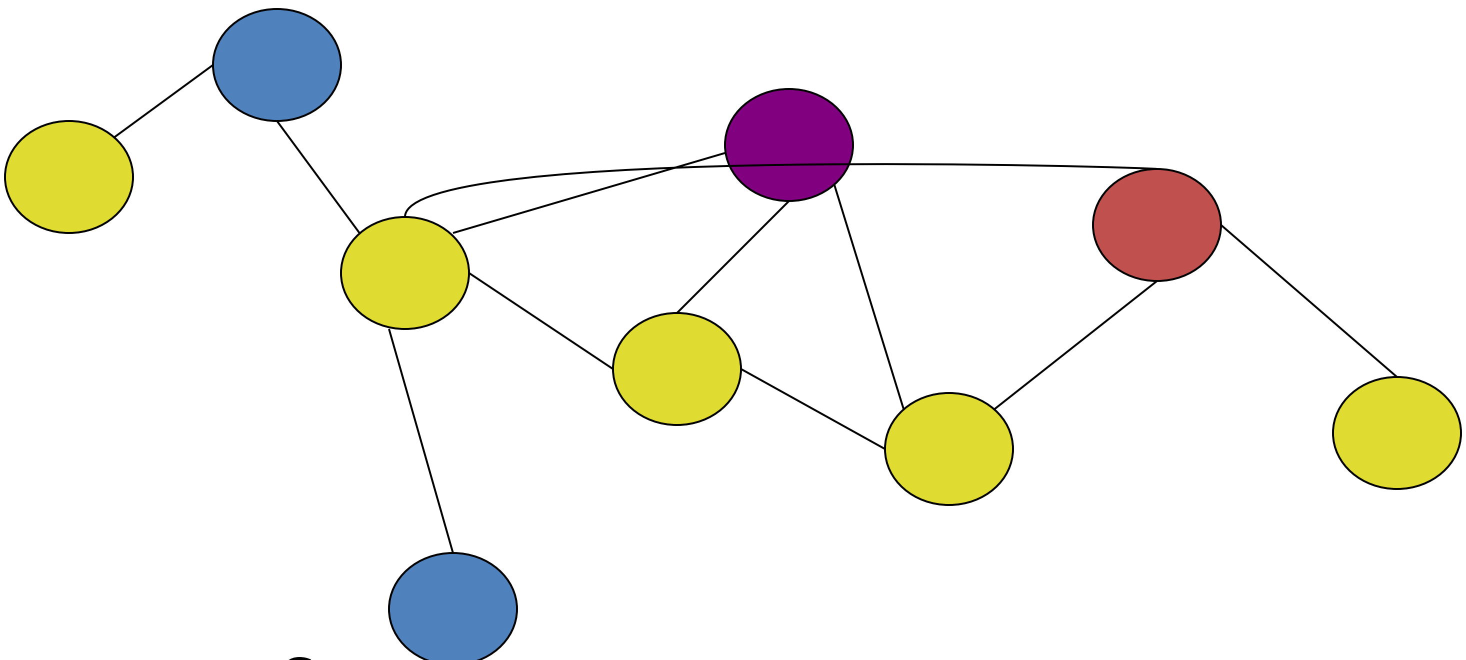


Replication Research

- How many replicas?
  - too many - waste of resources
  - too few - application suffers
- Leverage prior work!

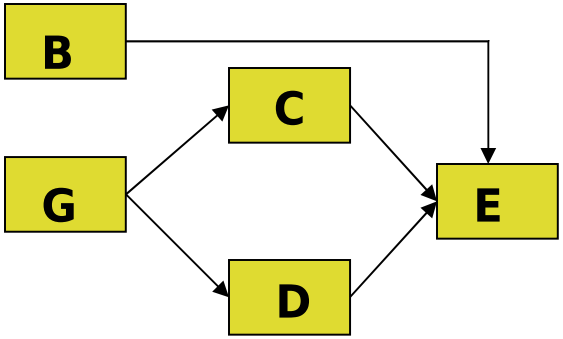
Client Replication

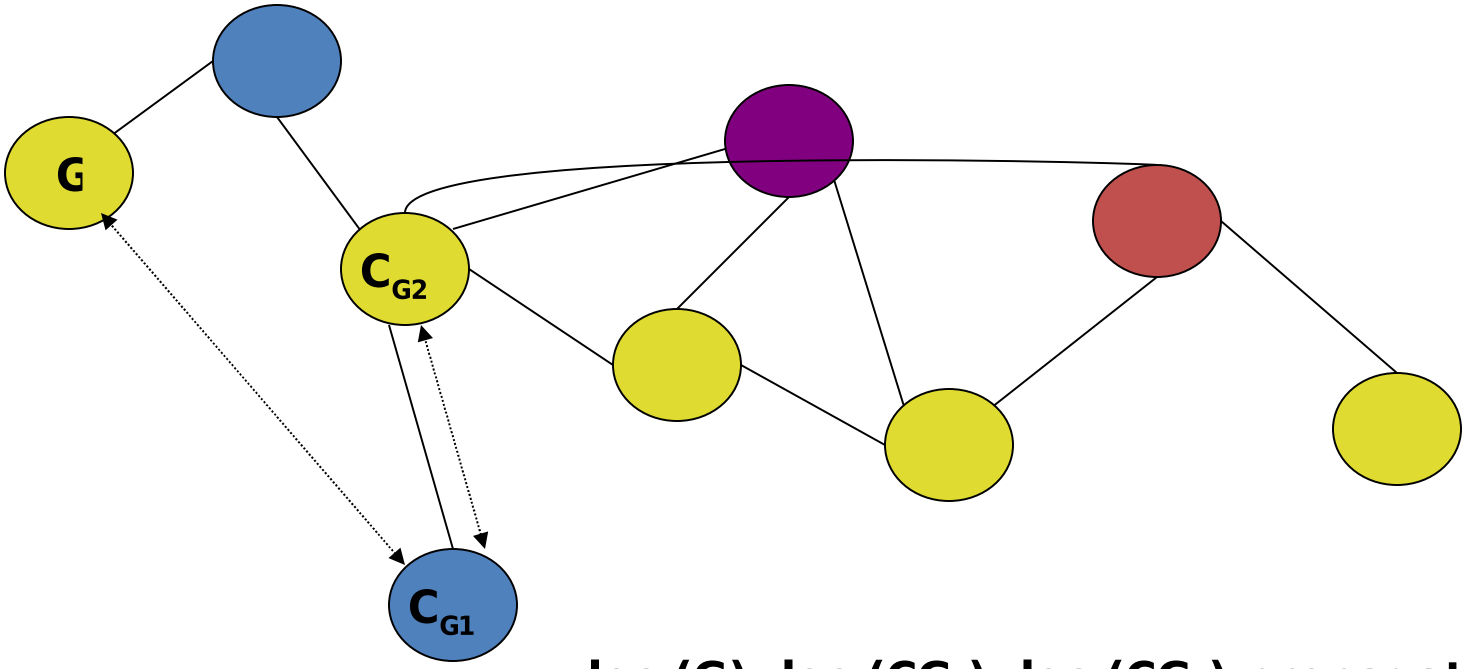




G

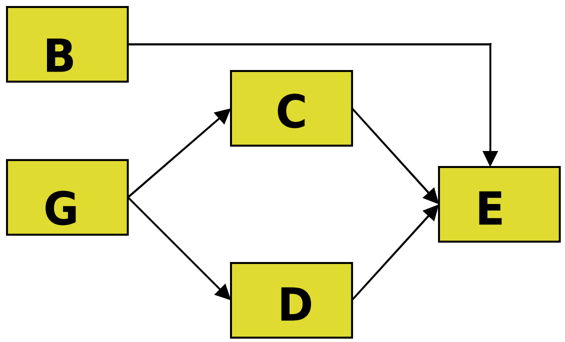
Client Replication

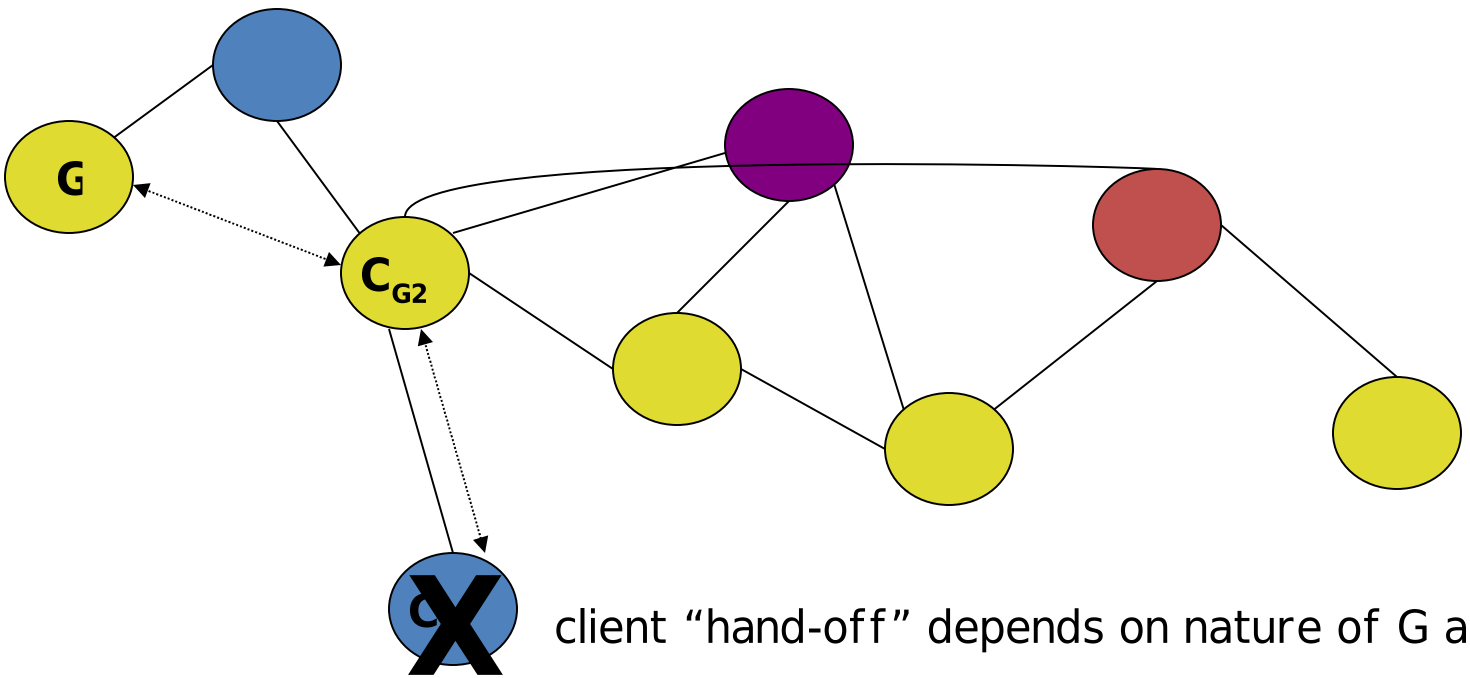




loc (G), loc (CG_1), loc (CG_2) propagated

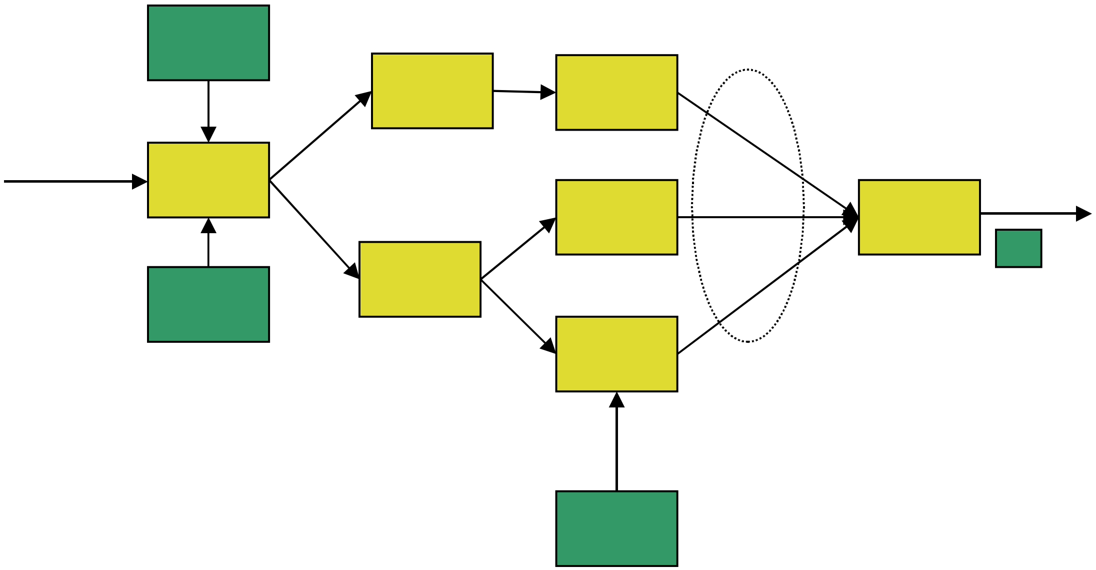
Client Replication





client "hand-off" depends on nature of G and interaction

Application Models



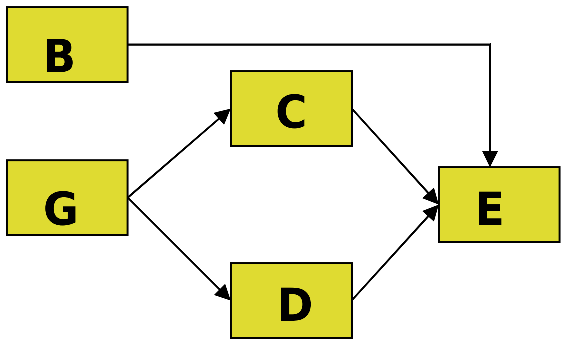
Reliability
Control dependence
Collective metrics

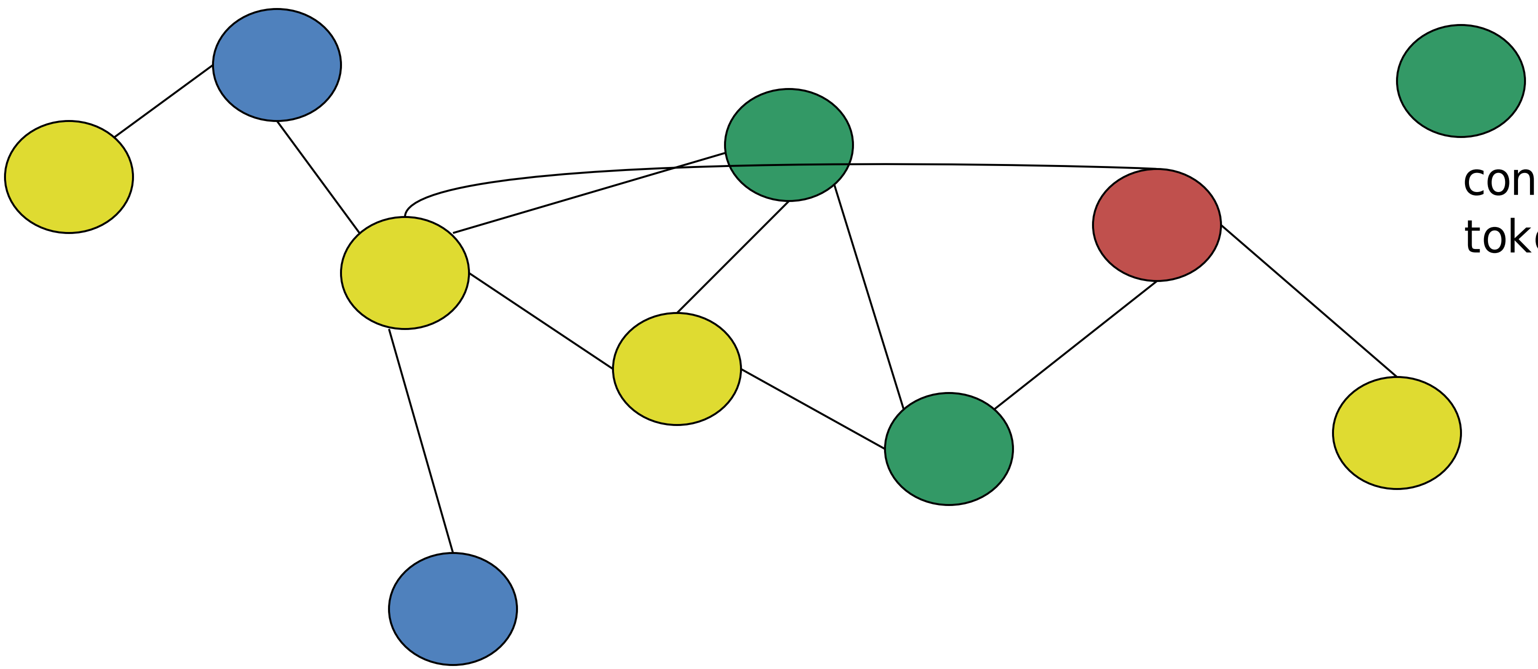
The Problem

- How to enable decentralized control?
  - propagate downstream graph stages
  - perform distributed synchronization

- Idea:
  - distributed dataflow - token matching
  - graph forwarding, futures (Mentat project)

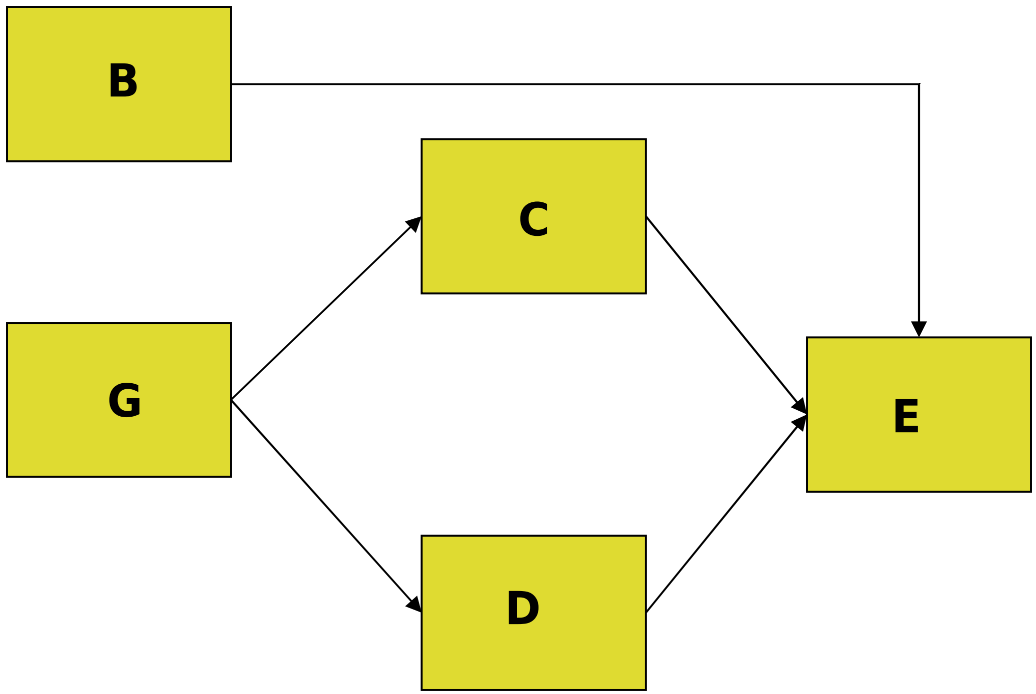
Control Example



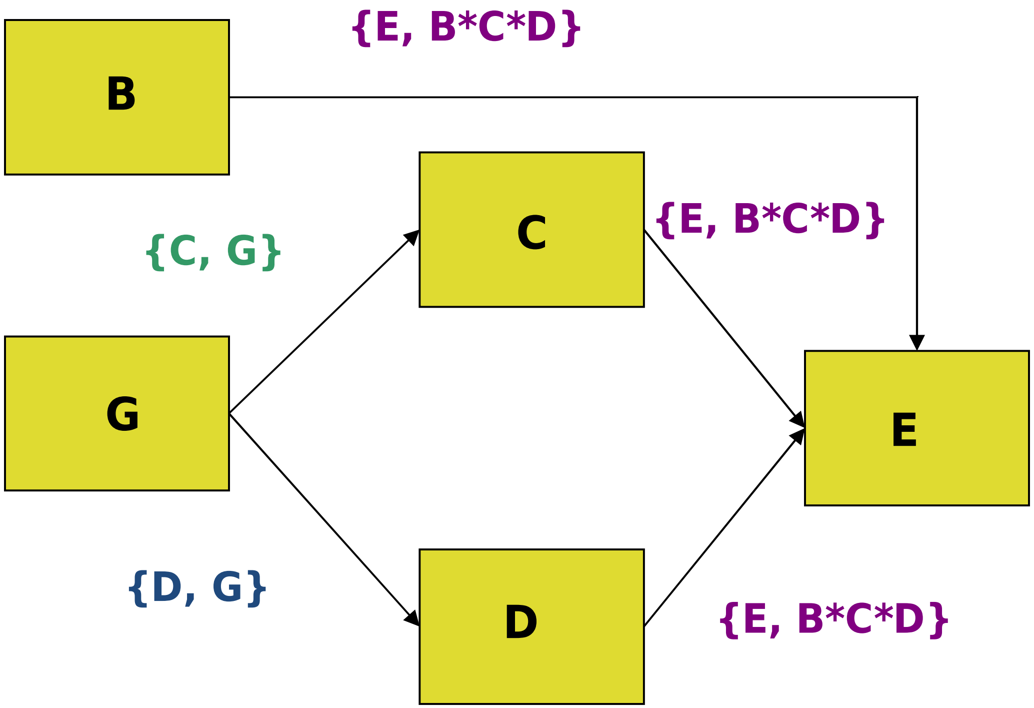


control node
token matching

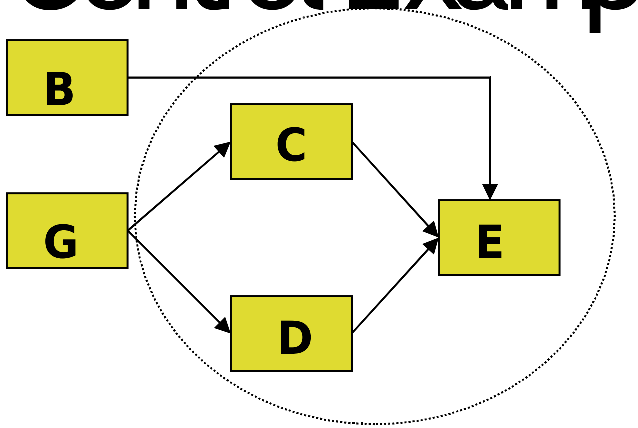
Simple Example

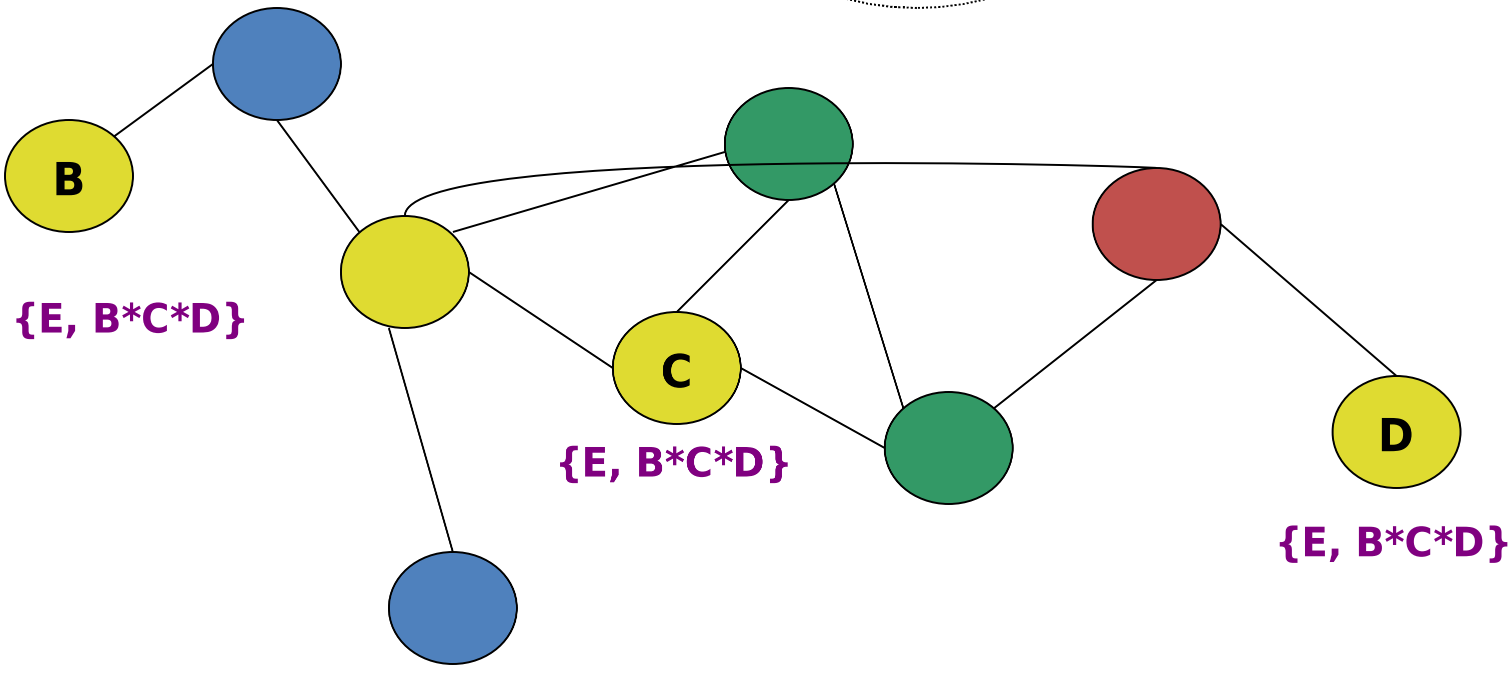


Control Example

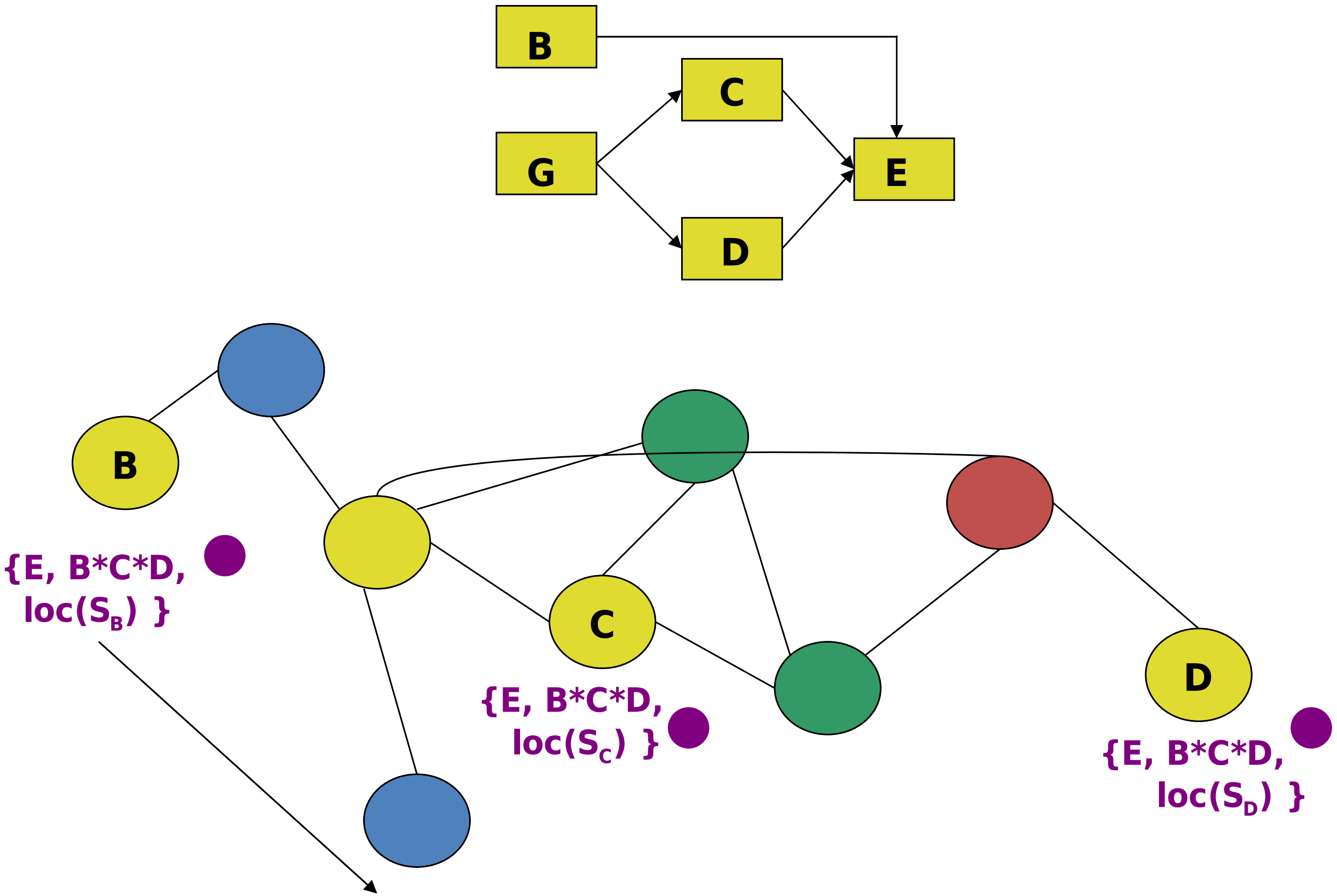


Control Example



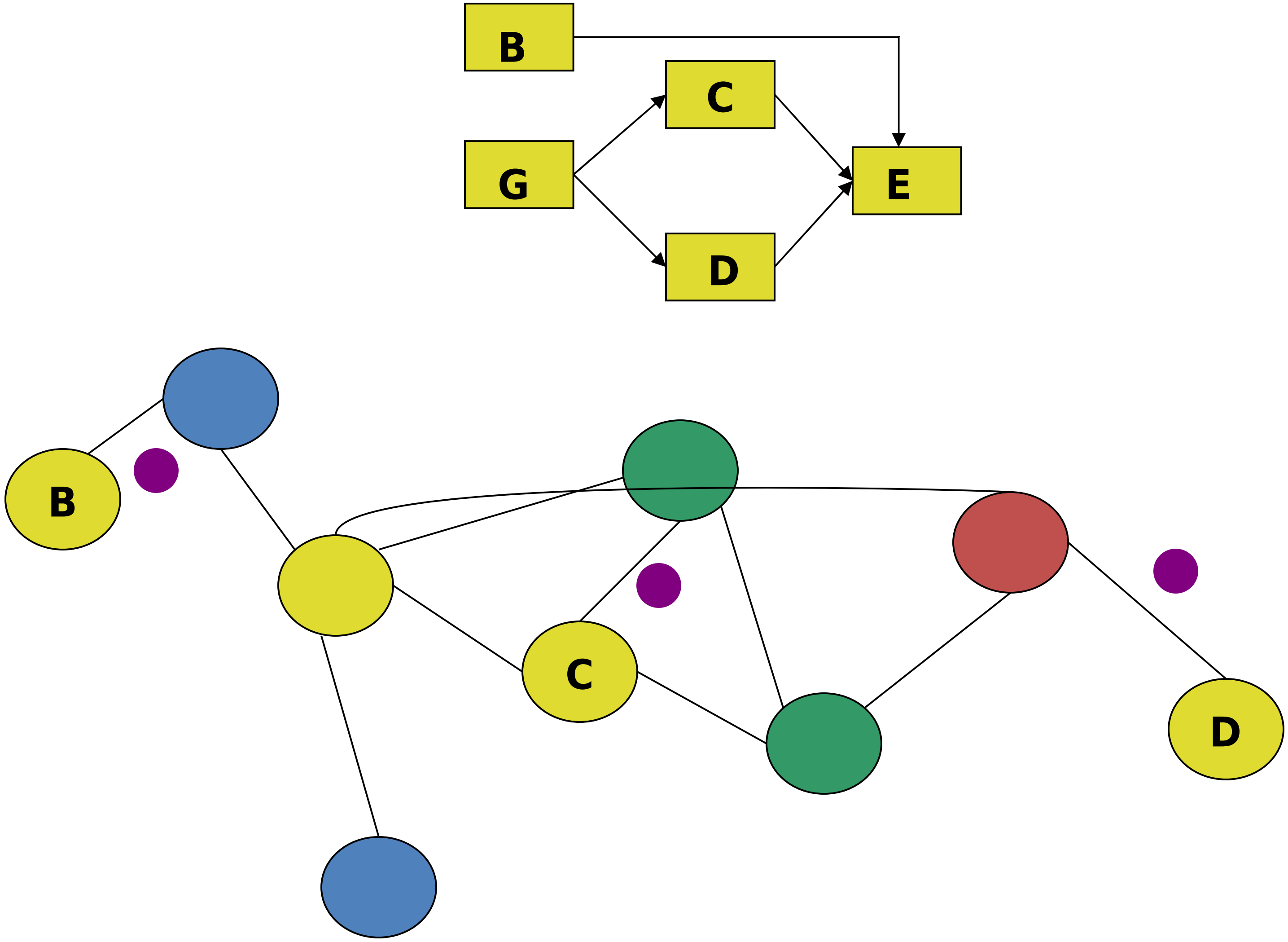


Control Example

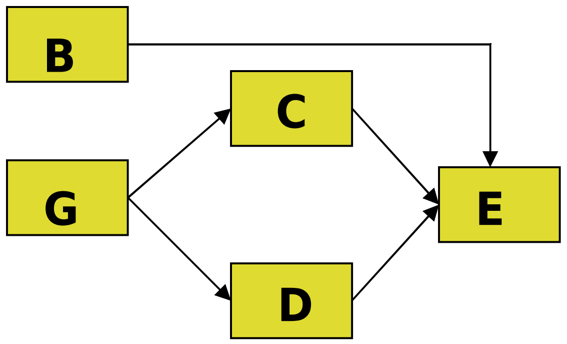


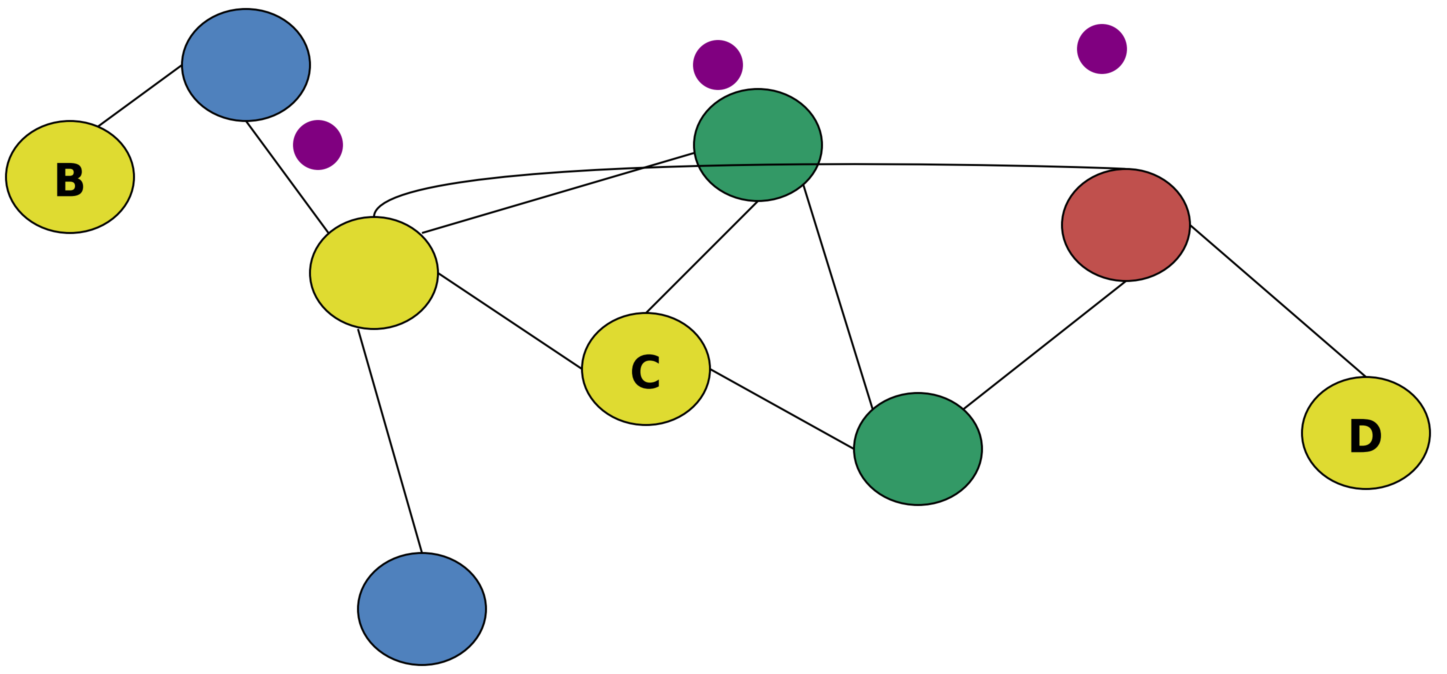
output stored at *loc*(…) – where **component** is run, or client, or a storage node

Control Example

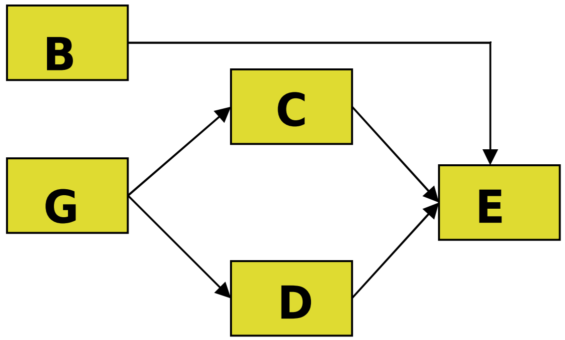


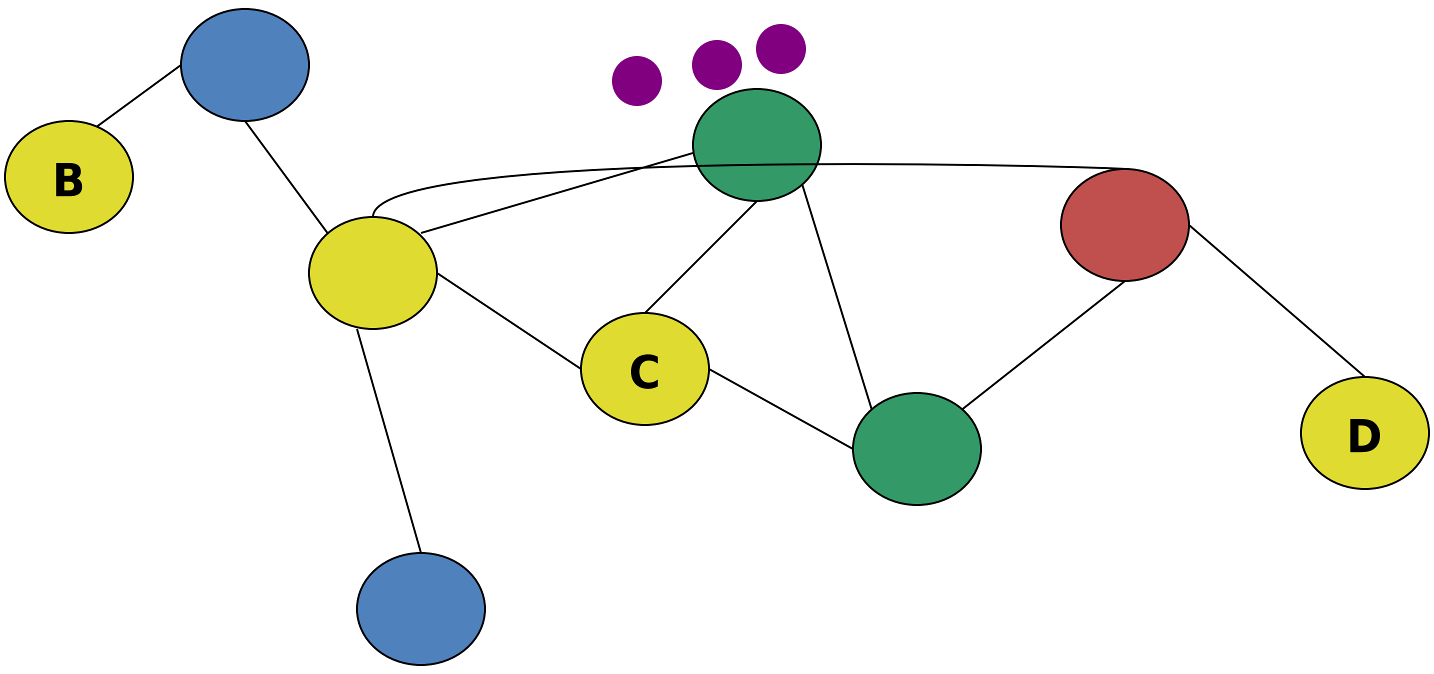
Control Example



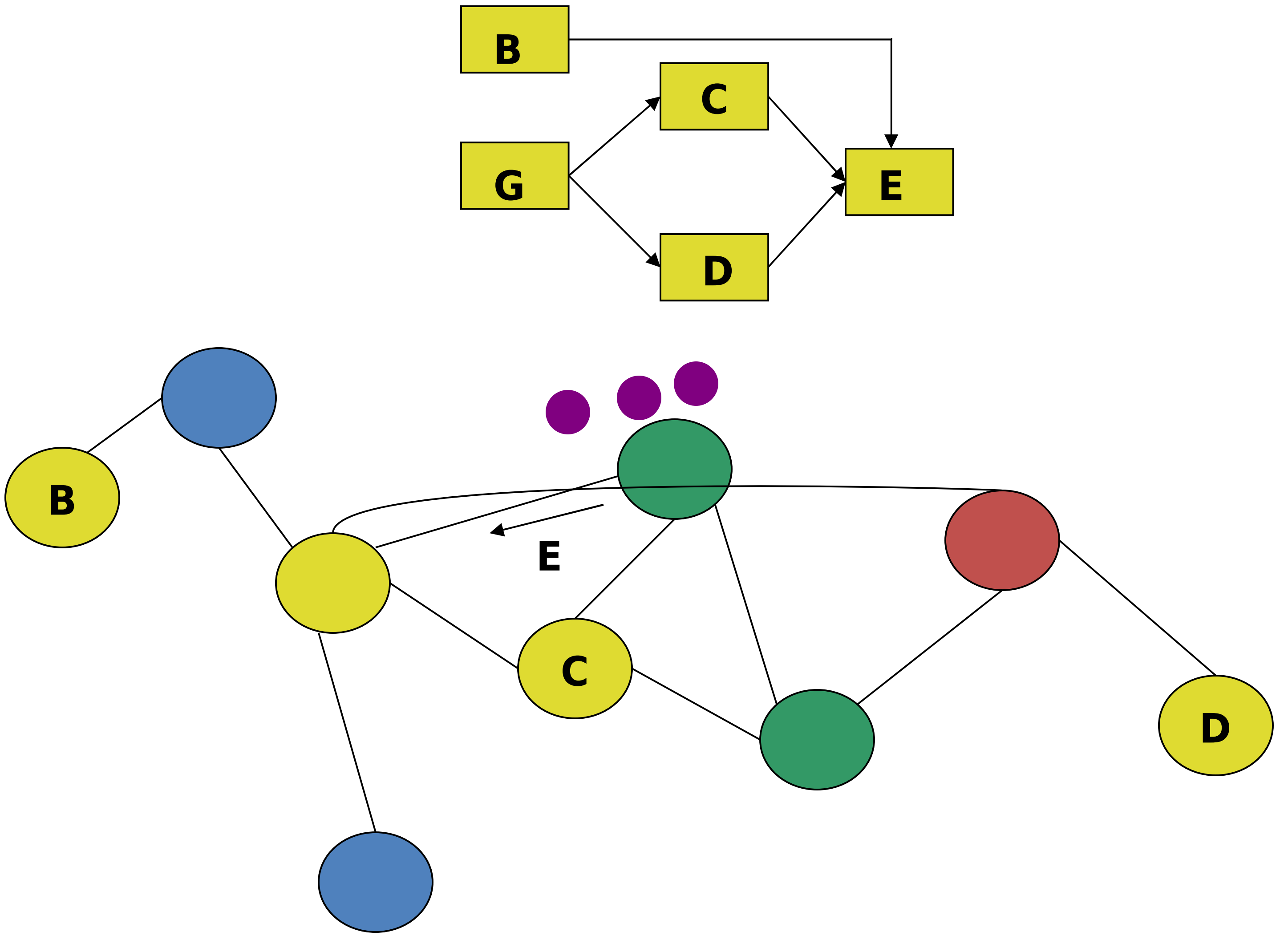


Control Example

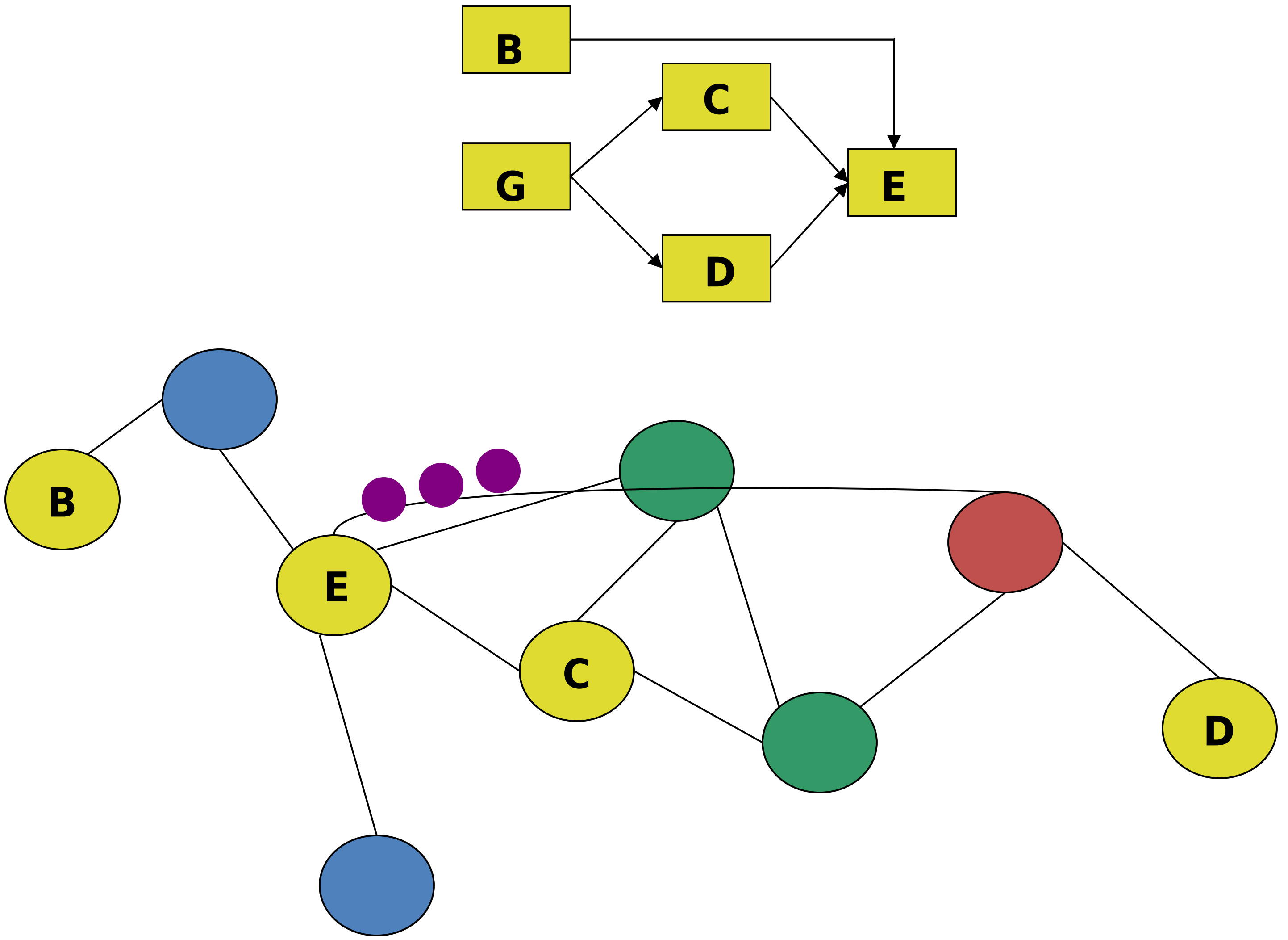




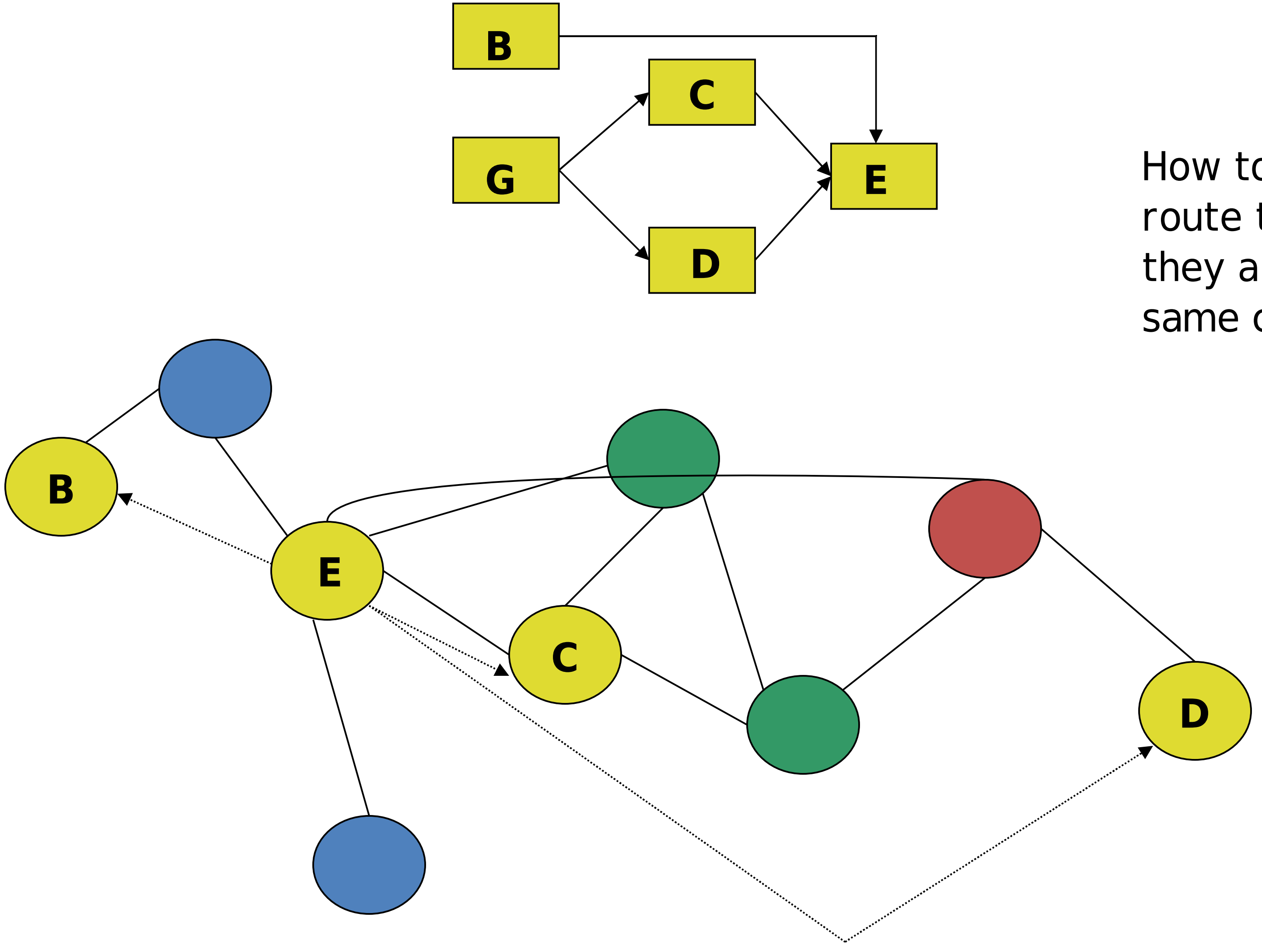
Control Example



Control Example

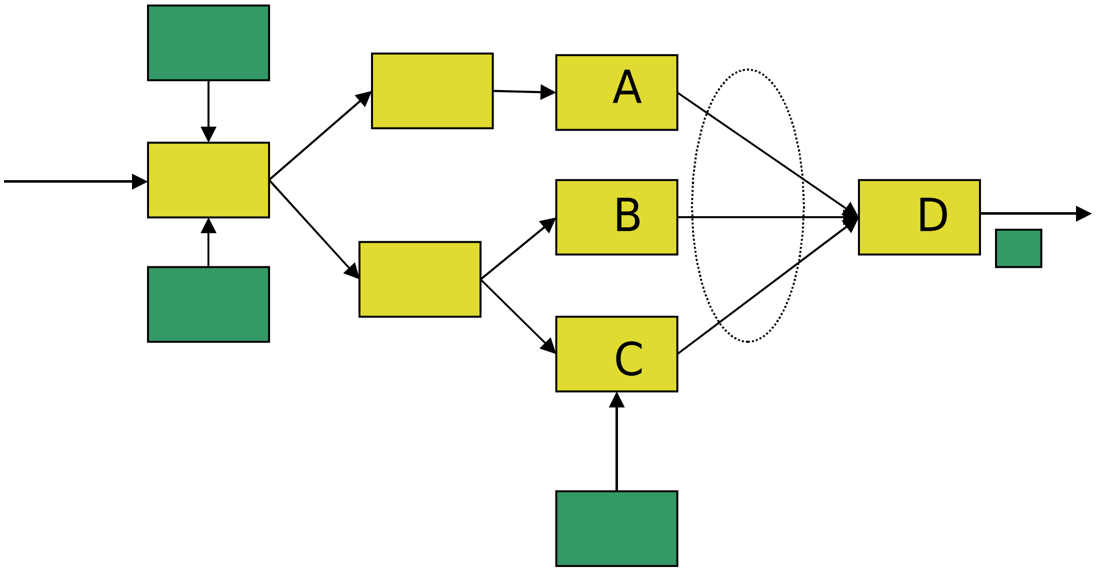


Control Example



How to color and route tokens so that they arrive to the same control node?

Application Models

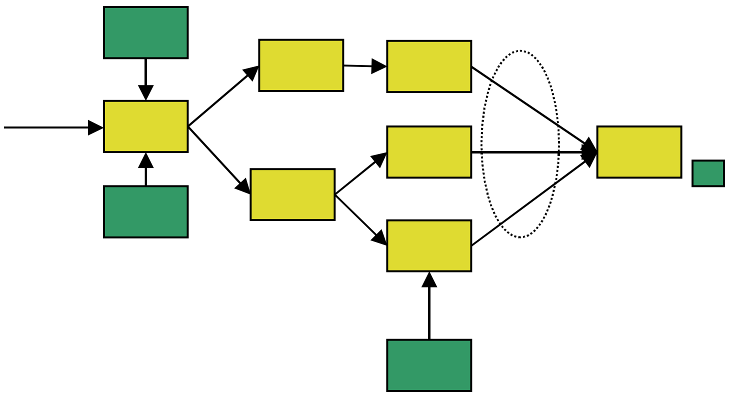


Reliability
Control dependence
Collective metrics

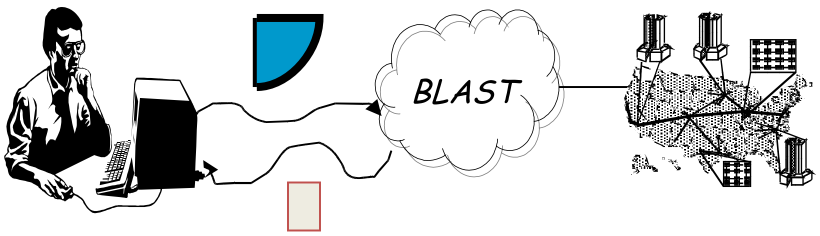
Collective Metrics

- Throughput not always the best metric
- Response, completion time, application-centric

– makespan



- response



Open Problems

- Support for Global Operations
  - troubleshooting - what happened?
  - monitoring - application progress?
  - cleanup - application died, cleanup state
- Load balance across different applications
  - routing to guarantee dispersion

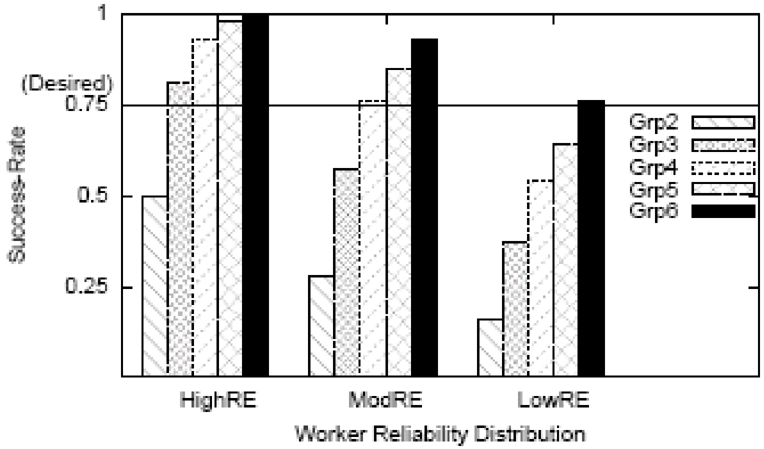
Summary

- Taming the dynamism of open distributed systems is a challenging endeavor
  - Grids "push the envelope" in this respect
- Grids can offer richer use cases to drive this research agenda
  - data X computation X distribution X heterogeneity
- Grids can learn a great deal from other communities
  - P2P, network systems

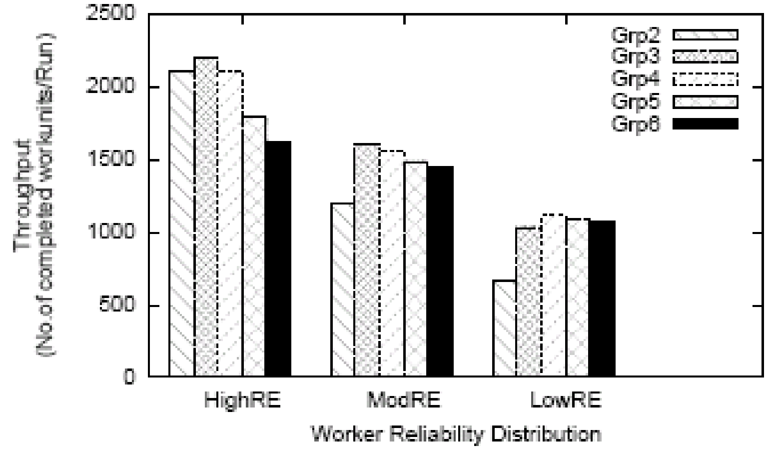
Questions

EXTRAS

EXTRAS - OG

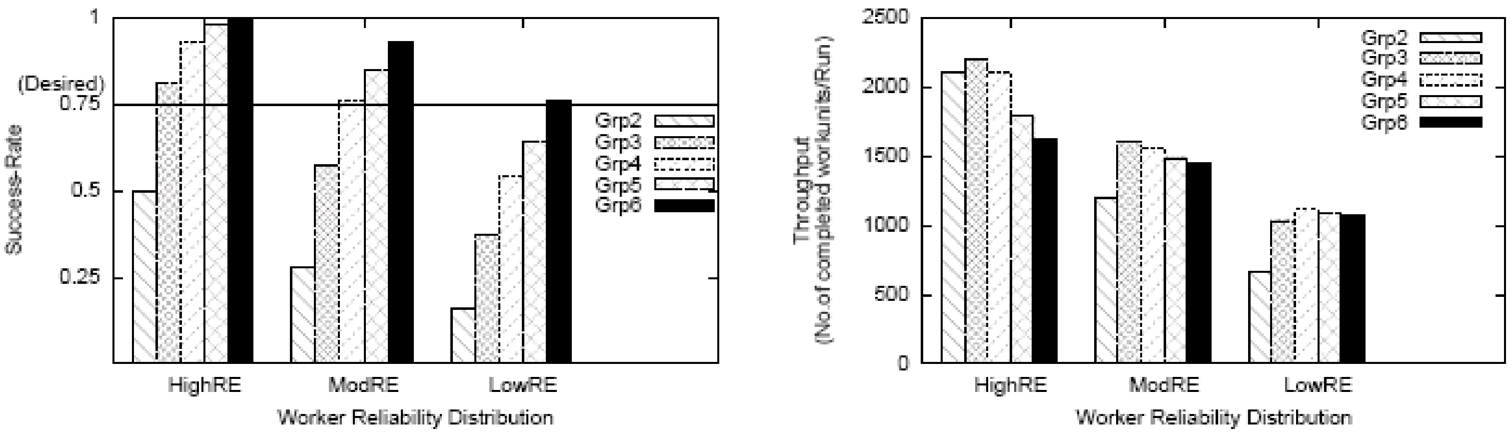


(a) Success-Rate



(b) Throughput

EXTRAS

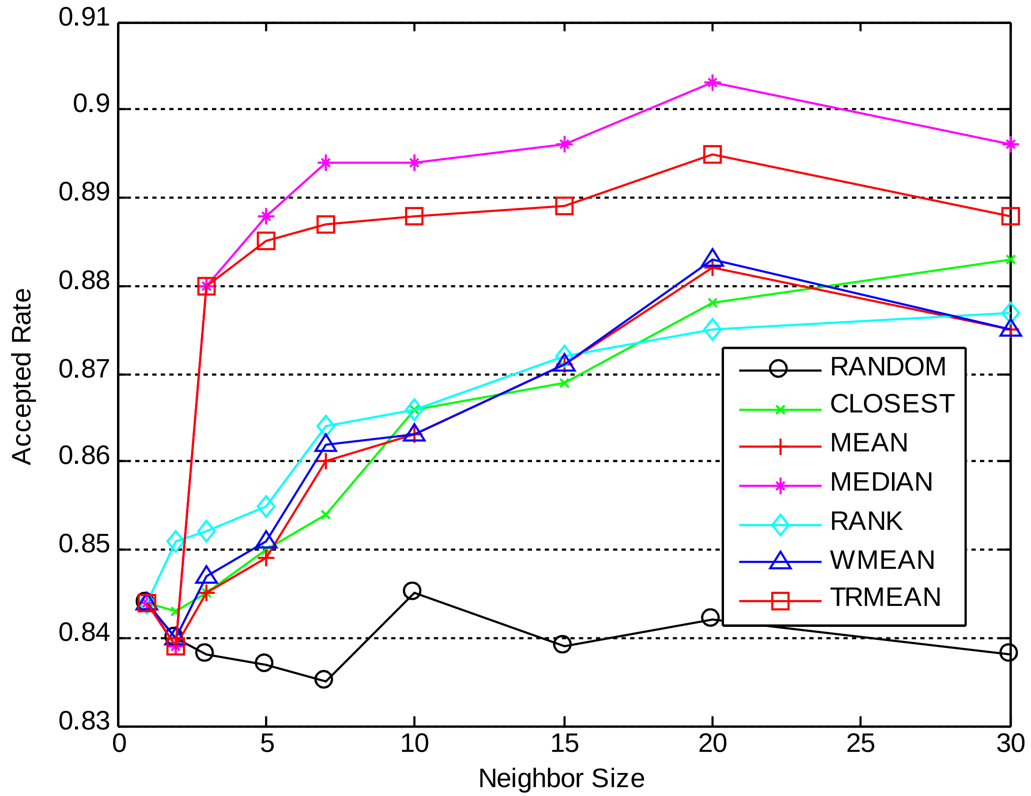


(a) Success-Rate

(b) Throughput

EXTRA - COMM

Combining Neighbors' Estimation

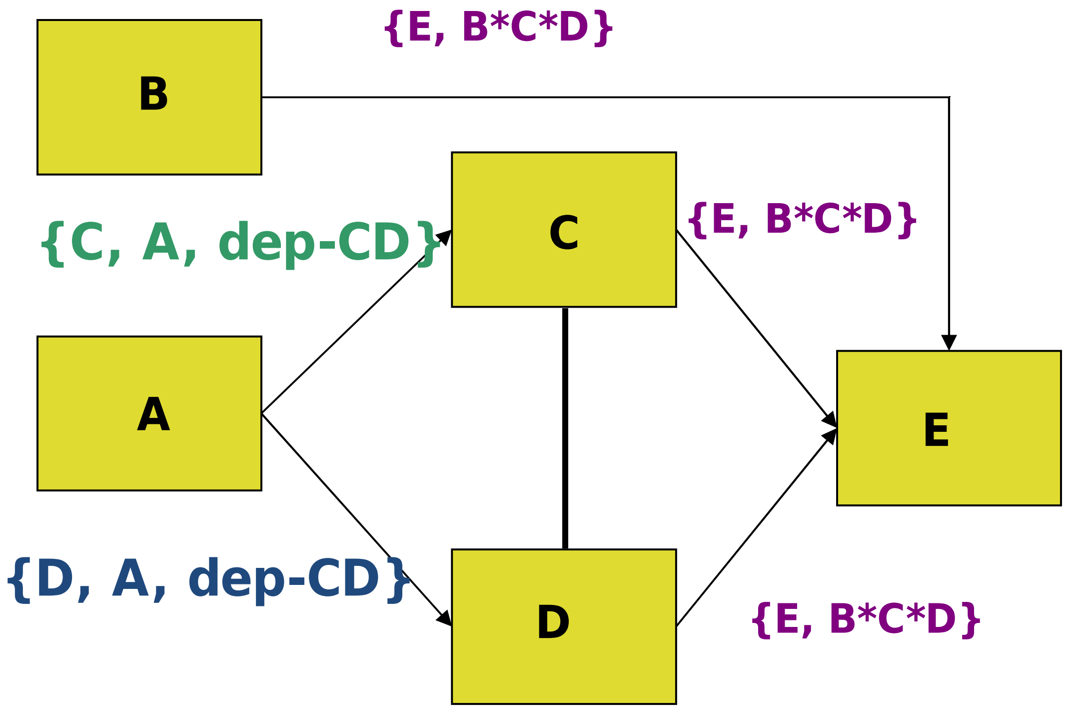


- MEDIAN shows best results – using 3 neighbors 88% of the time error is within 50% (variation in download times is a factor of 10-20)

RTT Inference

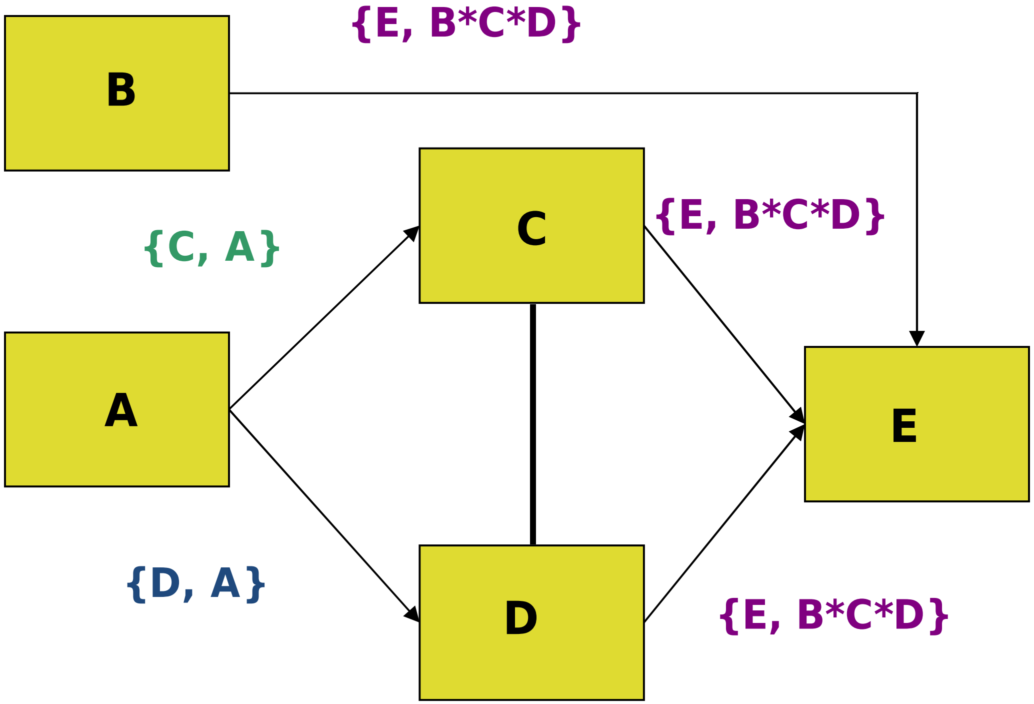
- >= 90-95% of Internet paths obey triangle inequality
  - RTT (a, c) <= RTT (a, b) + RTT (b, c)
  - RTT (server, c) <= RTT (server, n_i) + RTT (n_i, c)
  - upper- bound
  - lower-bound: |RTT (server, n_i) - RTT (n_i, c) |
- iterate over all neighbors to get max L, min U
- return mid-point

Other Constraints



C & D interact and they should be co-allocated, nearby …
Tokens in bold should route to same control point so a collective query for C & D can be issued

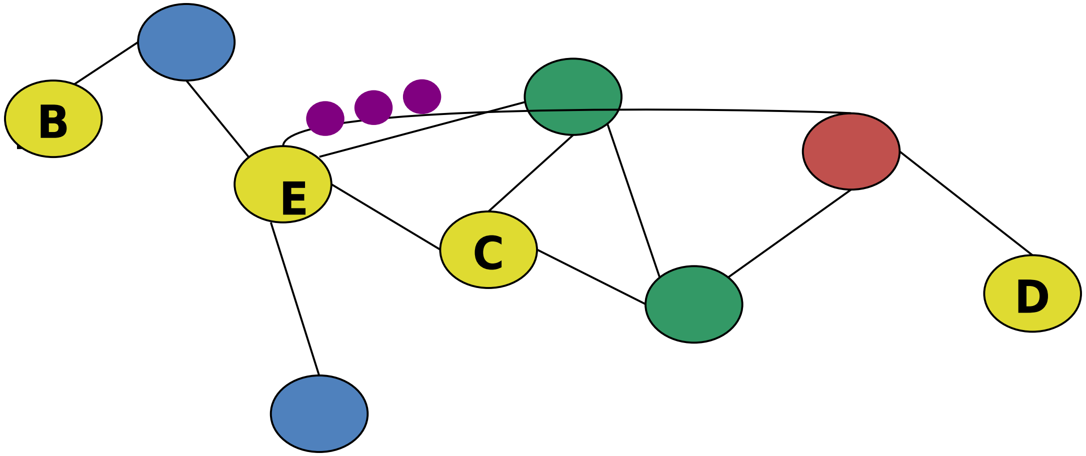
Other Constraints



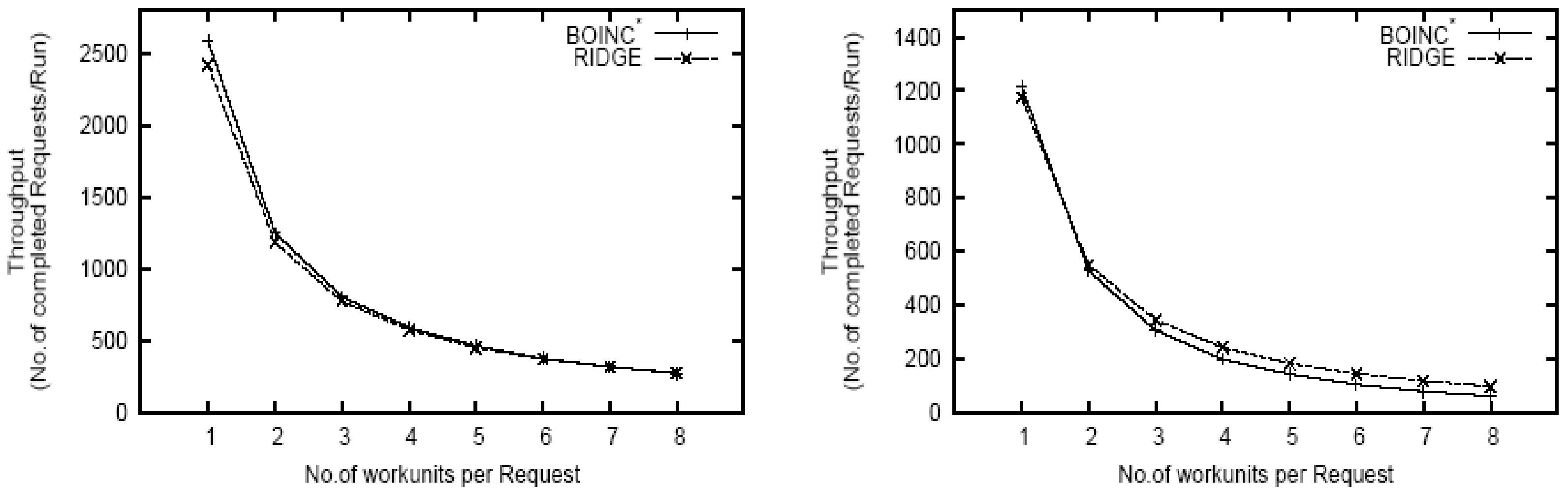
C & D interact and they should be co-allocated, nearby …

Token loss

- Between B and matcher; matcher and next stage
  - matcher must notify C_B when token arrives (pass loc(C_B) with B's token
  - destination (E) must notify C_B when token arrives (pass loc(C_B) with B's token



Throughput Comparison



(a) HighRE Throughput

(b) LowRE Throughput

Figure 12: Comparison of Request Throughput for different reliability environments

No loss in throughput