Simulations of Peer to peer resources management for Grids

Georges Da Costa

ISTI (Pisa) /UCY (Cyprus) / Coregrid

Plan

1. Introduction
2. Peer to peer Resources mAnager for Grids
3. Conclusion

Plan

1 Introduction

2 Peer to peer Resources mAnager for Grids

3 Conclusion

Context

Grids are difficult to manage
New trend : Plug several Grids together

Some problems

- Dynamicity : clusters come and go
- Administration : who is in charge
- Large scale, reactivity

What happens for a 10,000 nodes grid

Resource discovery & management

Large number of resources

- Web services
- Storage
- Desktop cycle stealing

A scheduler view for corporate grids

- Choose a place to run an application
- Lots of different resources : disk, cpu, network

$\longrightarrow$ How to manage these physical resources

Requests

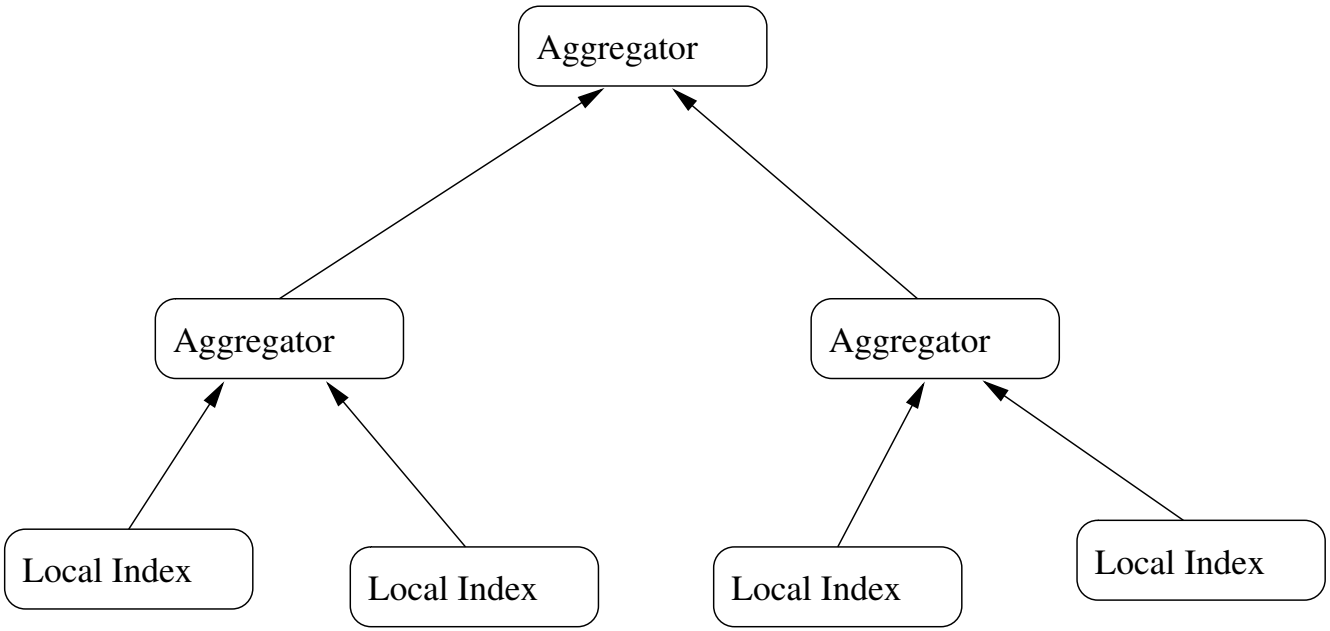
Example of request :32 freecpu and (ethernet or SCI)

Requests

Requests Get with multi-attribute with intervals

Updates are only done on precise values

Globus



Latence 10 min

Latence 1 min

Latence 1s

Timer based update

Globus limitations

The aggregator way

- Slow
- Local point of administration (political)
- Lots of work for the IT staff
- Single point of failure
- Local answering of requests
- No adaptation to data
- Timer based

Peer to Peer Systems

Pro

- Scalability
- Fully distributed
- Simple to manage

Con

- Precise request
- Only one answer

Plan

1 Introduction

2 Peer to peer Resources mAnager for Grids

3 Conclusion

Prag

We want a system that manage resources efficiently

Typical case of use : below the scheduler/user that want to execute a job

Efficient

- Few communication
- Small latency
- Good load balancing
- Scalable

Quality

- Number of answers
- Validity of answers

Environment model

Realistic data

- Academic grids or production one
- Which Grid
  - CoreGrid
  - Egee
  - French Laboratory (Cigri at Grenoble)

Current data

- Request Get : number of processor + stable data
- (work in progress) Model of the free processor number

Global hypothesis

Hypothesis

- Most of interval requests are opened
- There are different level of dynamicity (Upgrade/Get ratio)
- Some characteristics are constantly changing (bandwidth, load)

Requests

32+ freecpu and ethernet

Simplification

complex requests (and/or) are split and fusion is done on the initial node

Two type of resources

Dynamic

- Free Cpu
- Load

Stable

- Network infrastructure
- Total number of Cpu

Answering the requests

Two type of constraints

Dynamic resources

Optimizing the cost of update and the cost of requests

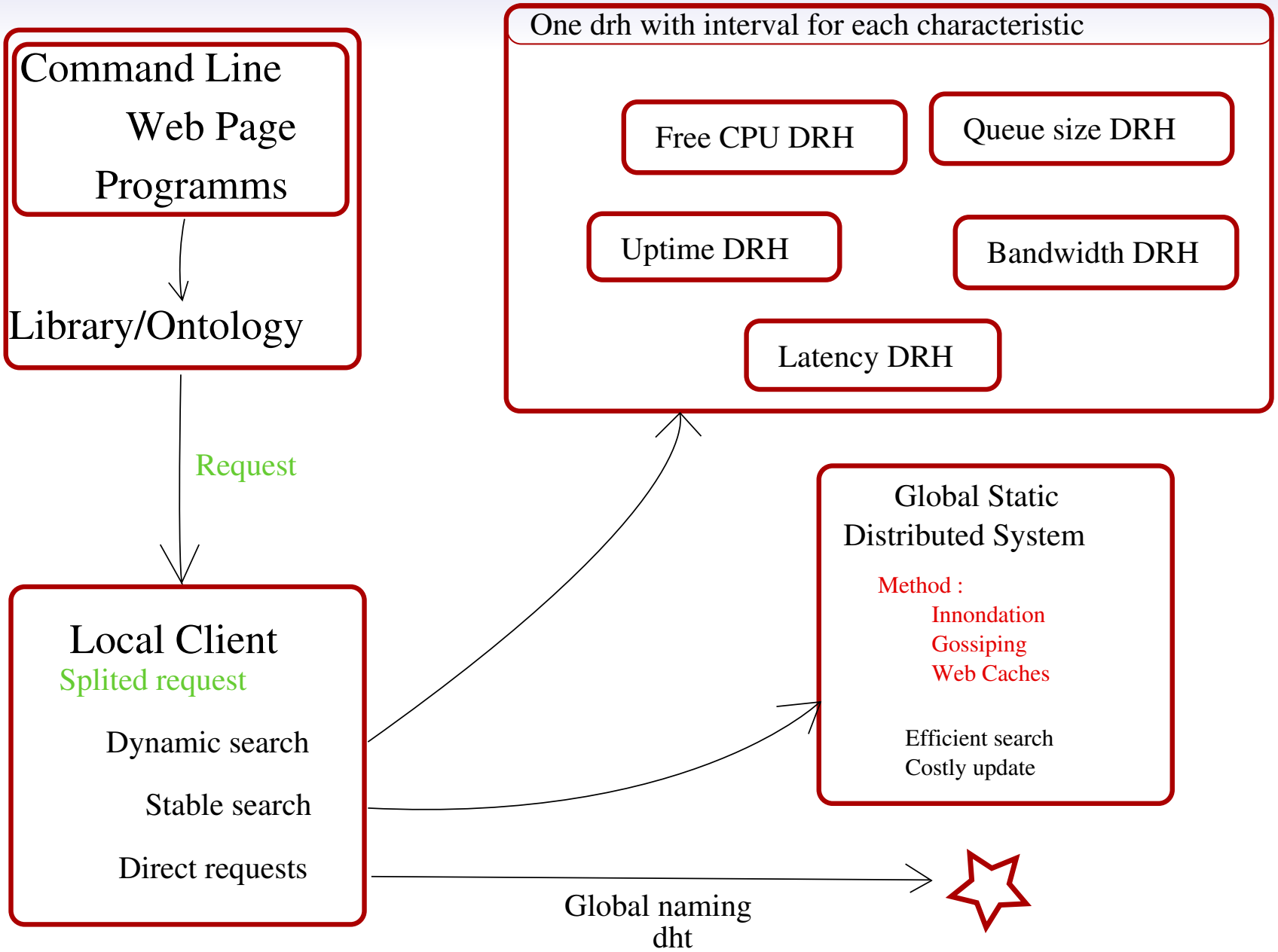
Stable resources

Optimizing the cost of requests

Efficient : A trade off between Update/Request cost

Then a global trade off : Efficient/Quality

Structure



Event based update

Distributed requests handler

Basic operations

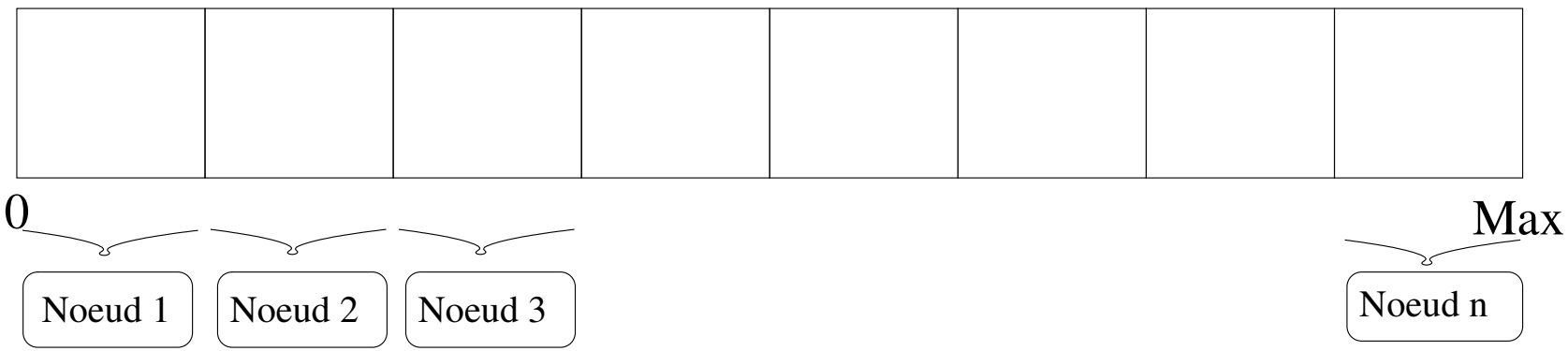
- data list = get($bound_1$, $bound_2$);
- data = get(key);
- update(key, data);

Metrics

- Number of messages for a request
- Number of messages for an update
- Number of messages for a crash/join/leave
- Workload balancing
- Percentage of answers

Current Peer to Peer Interval-enabled systems

	request a value	update a value	node join/leave
Probe	$N^{1/d}$	n/a	large
Nikos05	log(n)	log(n)	log(n)
Baton	log(n)	log(n)	log(n)
Ganesan04	log(n)	log(n)	log(n)



Problem : requests cost is log(n)+X

Problem : workload for requests is not well distributed

Most requests are open

Global hypothesis

Hypothesis

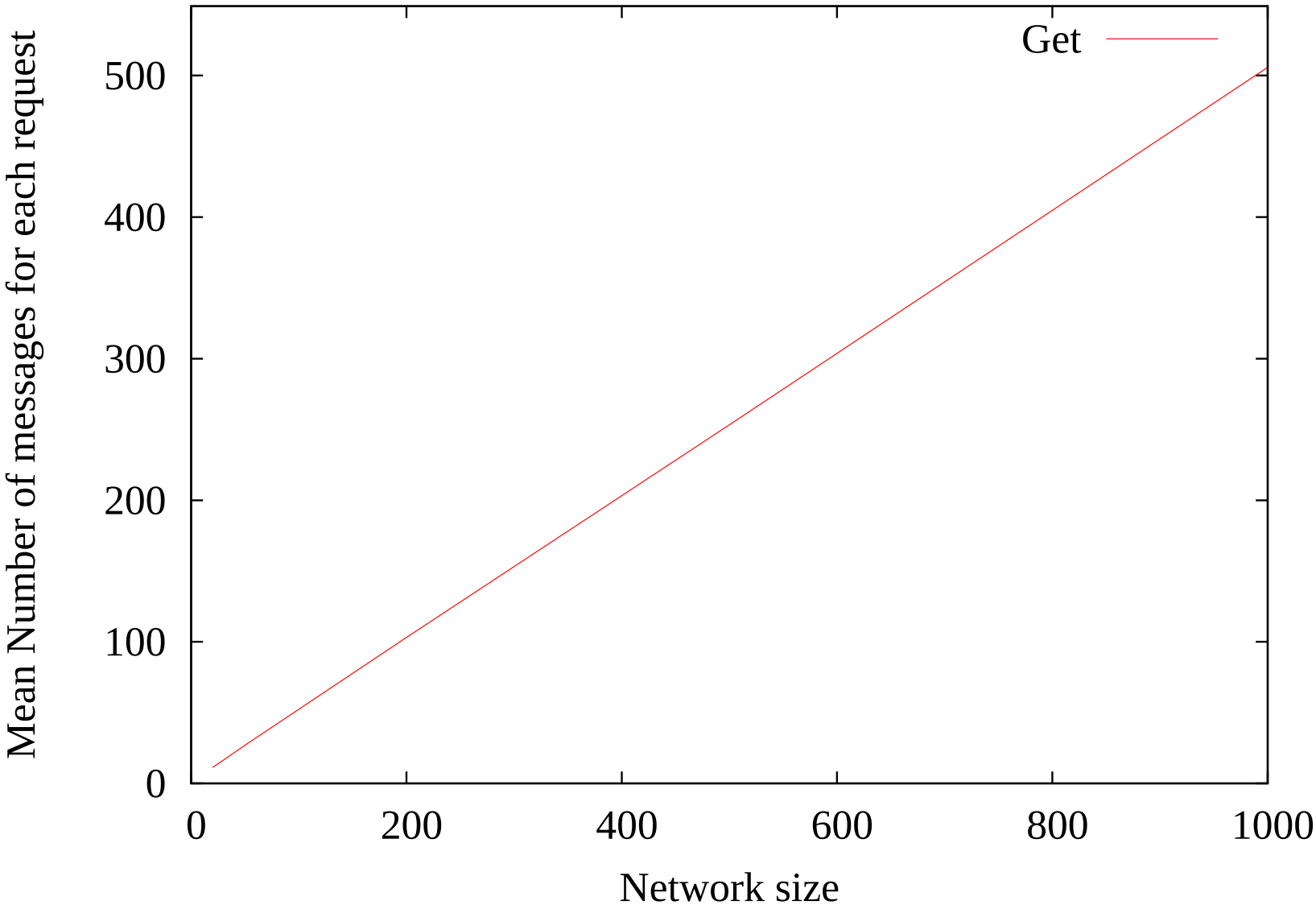
- Open intervals
- Uniform requests
- Time between requests is large compared to their treatment time

Test system

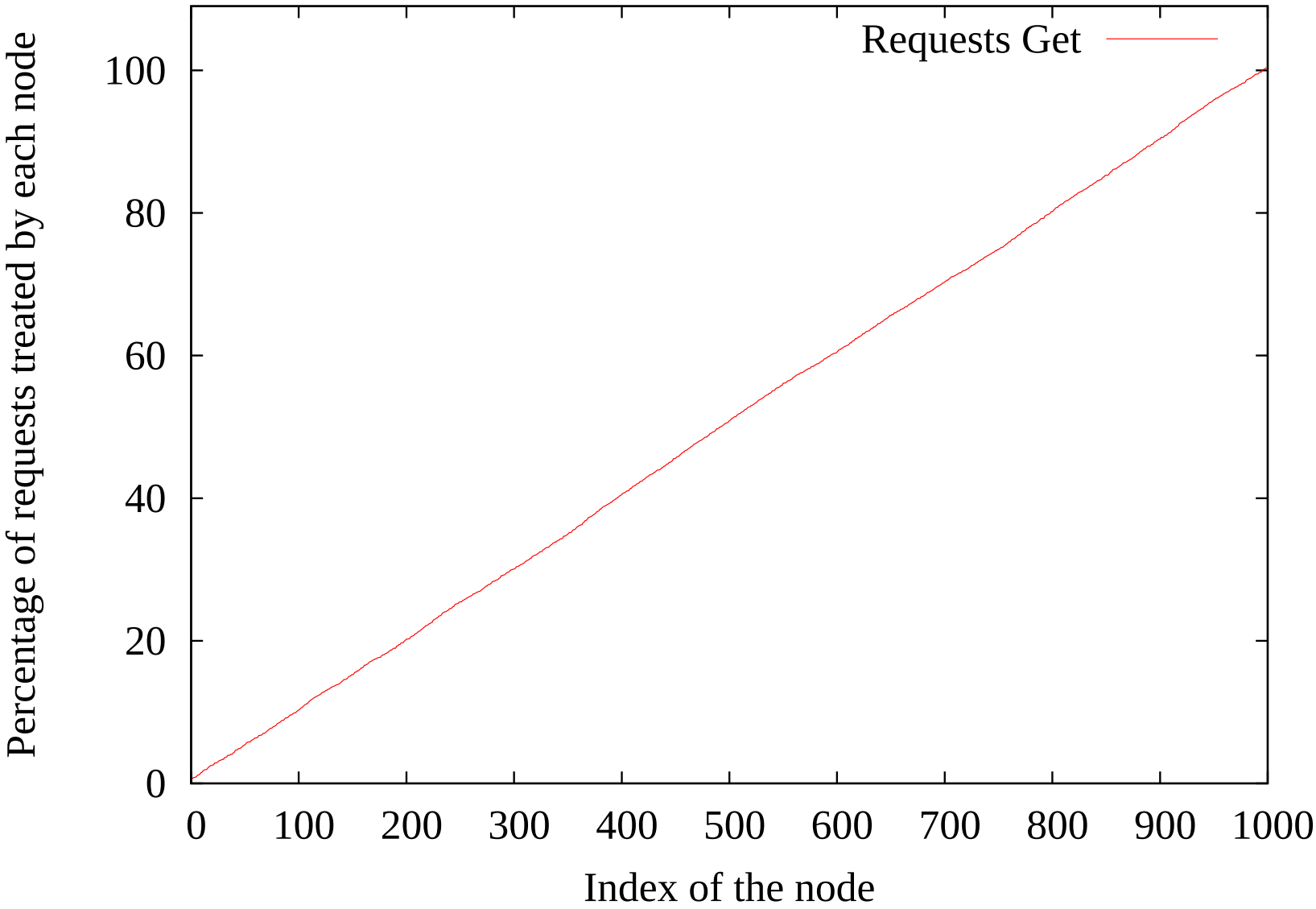
- Chord like
- Interval requests are forwarded from interval to interval (sequential requests)

10000 requests
1000 nodes

Request Cost in an standard Peer to Peer system



Workload Distribution in a standard Peer to Peer system



First experiments

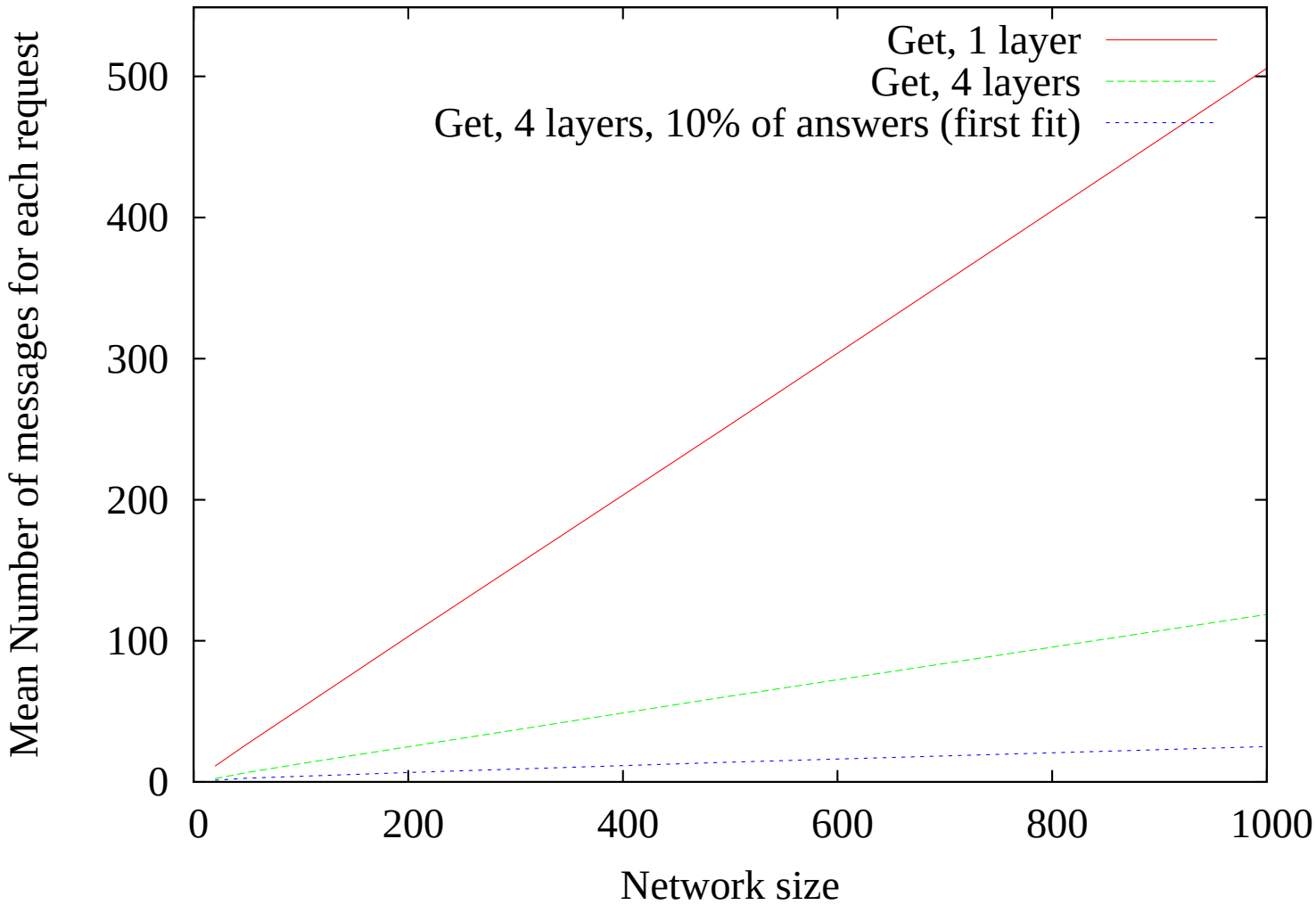
What append with several DHT

New test system

- 4 identical DHT
- Upgrade becomes 4 upgrades
- Request Get becomes one request on one randomly chosen DHT

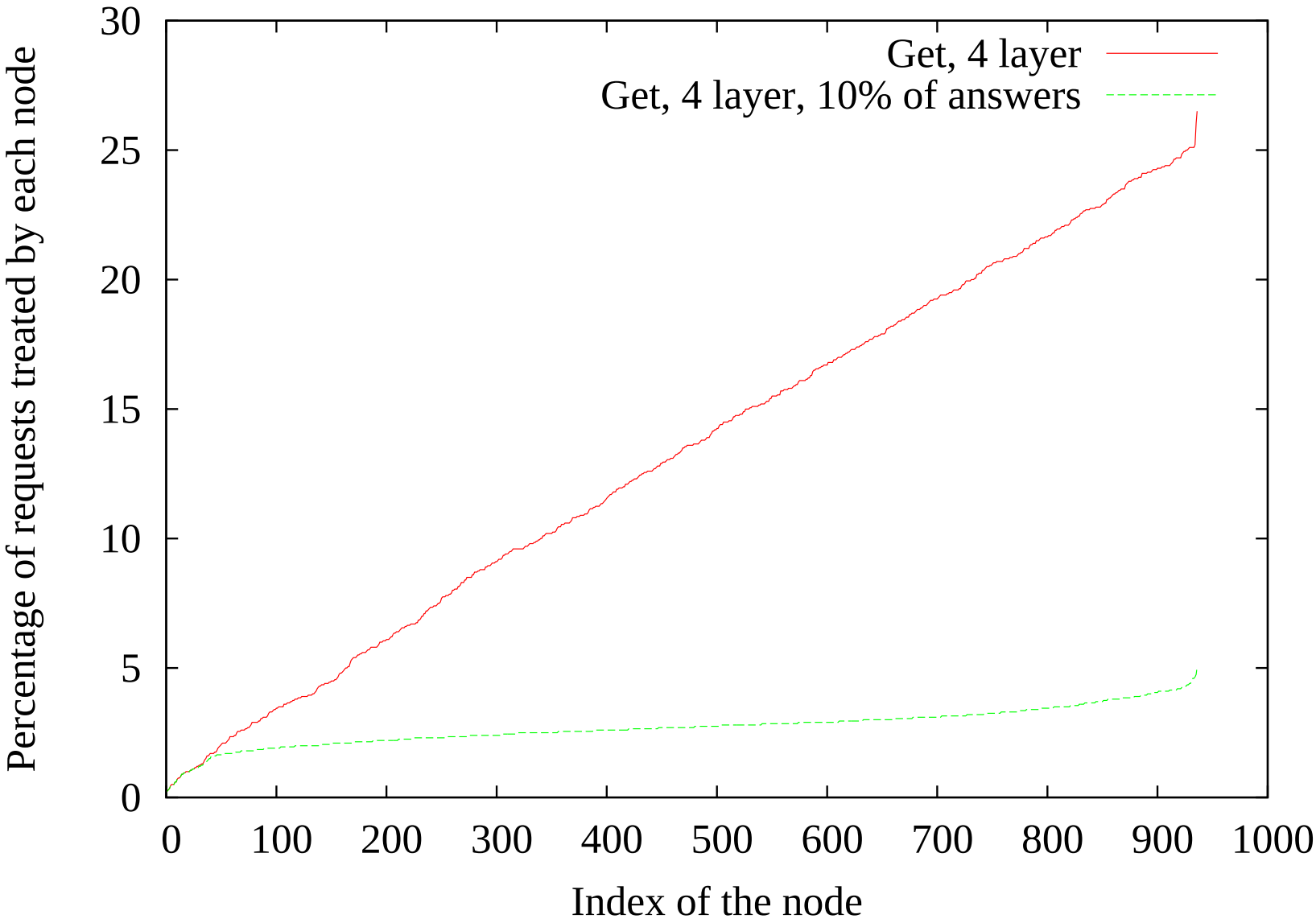
Upgrades are more costly, but Get are less

Request Cost



Better latency means better quality

Workload Distribution



Unification

How to choose the number of levels ?

A priori evaluation of the level number in function of the ration Get/upgrade ?

Problems

- The ratio can change
- Difficult to have him a priori

$\Longrightarrow$ Make this number adapt to the ratio

- 1 Dht $\longrightarrow$ standard case
- n Dht $\longrightarrow$ total duplication

(n is the number of nodes in the system)

Unification

Basic operation

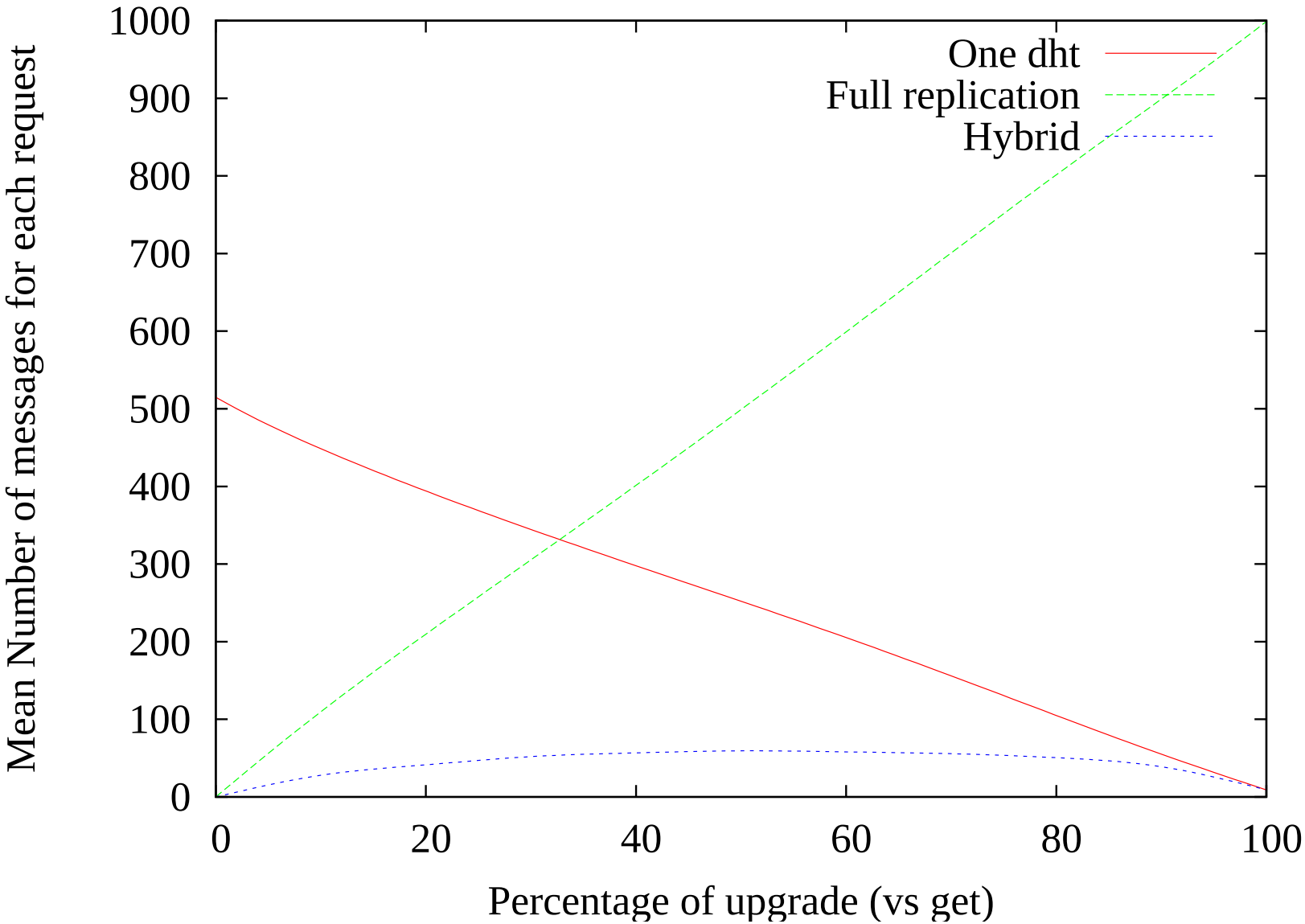
- Fusion (p DHT become $p-1$)
- Split (p DHT become $p+1$)

Node responsible of the value 0 decide

Other nodes send them their values

Find a local minimum of the number of messages

Unification



Plan

1 Introduction

2 Peer to peer Resources mAnager for Grids

3 Conclusion

Conclusion

Conclusion

- Globus can be improved for large grids
- We propose a 'self*' architecture to manage resources management for large grids

Perspectives

- Model of the resources
- Simulation of several P2P range query systems
- Integrate the operation p DHT $\rightarrow p+1$ DHT
- Define the tradeoff Quality/Efficiency
- Integrate Multi-Attribute at the query level

Questions ?