GWiQ-P: An Efficient, Decentralized Grid-Wide Quota Enforcement Protocol

Kfir Karmon, Liran Liss and Assaf Schuster
Technion – Israel Institute of Technology

SYSTOR 2007
IBM HRL, Haifa, Israel
Background – Grid Resources

Sum of all resources of the same type constitutes a Grid Wide Resource.

Same resource type scattered over the grid.

Grid Wide Resource
Background – Grid Resources

- Grid Wide Resources
  - CPU hours
  - Disk space
  - DB Connections
  - Outbound traffic
  - Concurrent number of CPUs
  - Allocated RAM
  - Floating Software Licenses
  - Open sockets
  - Etc…
GWiQ Motivation

- A grid wide resource tends to be huge and can be exploited
- **Grid Wide Quota Enforcement** is vital:
  - **Security**: Prohibit malicious use
  - **Fail Safe**: Prevent resource leaks (bugs)
  - **Financial**: Moderate use per paid share
Centralized GWiQ Enforcement

- Central server holds the GWiQ bounds for each (user, resource) tuple
- Per request, resource usage permits are leased until the GWiQ is exhausted.
Objectsives

- We strive for a **Grid Wide Quota** enforcement protocol that is:
  - **Decentralized**: No hotspots, No single point of failure.
  - **Efficient**: Overcome latency caused by grid’s physical distribution.
  - **Scalable**: Can handle Mega-Grids
GWiQ-P: Grid Wide Quota enforcement Protocol
GWiQ-P: Basic Concept

GWiQ Enforcement

At all times the sum of all local quotas < GWiQ

- Using sandboxes to enforce local quotas
- Given an attempt to access the resource:
  - If (local-quota >= request) then
    - Grant access
    - local-quota = local-quota - request
  - Else
    - Freeze job execution until local-quota reinforced
A resource coin denotes the smallest consumable portion of a grid resource.

Each (user, resource) GWiQ is broken down to coins.

A user’s job may use the resource up to the amount that the coins are worth.

i.e. Depositing four 1MB coins grants the job (another) 4MB to use.

Local Quota = Hosting SBox’s resident coins
GWiQ-P – Spanning Forest

- Using a BF-based alg we build a spanning forest.
- A sandbox hosting a needy job will start forming a tree around itself.
- At all times, each neighbor will join the tree to which it is closest to its root.
  - Member of one tree at a time.
- Surplus coins will be transferred to the root.
GWiQ-P : In action 1/5

1

2

3

I need 1 coin

4

5

6

7

8

I need 1 coins

9

10

11

12

I need coins

13

14

15

16
GWiQ-P: In action 2/5

1

2

3

I need 1 coins

4

5

6

7

8

9

10

I need coins

11

12

13

14

15

16
GWiQ-P: In action 3/5

1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16

I need 1 coins
I need coins
I need coins
GWiQ-P : In action 5/5

I need coins
GWiQ-P: Fault Intolerance

- Links and nodes may crash
  - Link crash → Passing coins are lost
  - Node crash → Local coins are lost
- Lost coins ⇔ Unfair GWiQ reduction
- Can’t be sure how many coins are on the wire
  - Did my neighbor send something?
  - Have the coins I sent reached already?
- How do we restore unknown lost coins???
FT Solution 1: Transactional

- Coin transfers are done in Transactions
- Pros
  - Coins never lost on transfer
- Cons
  - Slowdown
  - Needs persistent storage
  - Node failure (temporarily?) reduces GWiQ by local quota
Machine down $\iff$ All of its links are down

Each node holds a counter for every link.

- $\text{FTCounter} = \text{Sent coins} - \text{Received coins}$
- $\text{FTCounter} = \text{Net number of transferred coins}$

When a link goes down:

- If $\text{FTCounter} < 0$
  - Create a fictive demand for $|\text{FTCounter}|$ coins
- Else
  - $|\text{FTCounter}|$ coins are added to the node’s surplus
FT Solution2: Light Weight 2/6

Example 1:
Example 2:

- FTCounter=+1
- FTCounter=-1
- I need 2 coins
- FTCounter=+3
- FTCounter=-1
- I need 2 coins
- Temporary breach
- Fictive Demand
- I need 1 coins
- I need 2 coins
Example 3:
FT Solution2: Light Weight 5/6

Example 4:

- FTCounter=0
- FTCounter=0
- FTCounter=0
- FTCounter=0

- FTCounter=+2
- FTCounter=-2
- FTCounter=+2
- FTCounter=-2

- Temporary breach
- Fictive Demand
  I need 2 coins
**FT Solution2: Light Weight 6/6**

- **Pro**
  - No latency
  - No need for persistent storage
  - GWiQ never unfairly reduced

- **Con**
  - May introduce temporary GWiQ breaches
FT Solution3: Hybrid 1/2

- Use the Light-Weight solution regularly
- Only con to deal with: GWiQ breaches
  - Caused by loaded FTCounters
- Issue FTCounter balancing Transactions
  - If $|\text{FTCounter}_{i,j}| > \text{Threshold}$
    - Disallow link usage. Start transaction.
    - $\text{FTCounter}_{i,j} = \text{FTCounter}_{j,i} = 0$
    - End transaction. Resume link usage.
  - Or, issue periodically.
FT Solution3: Hybrid 2/2

- **Pro**
  
  Due to Transactions:
  - Reduce breaches’ size
  - No slowdowns (mostly)

  Due to Light Weight:
  - No persistence storage (mostly)
  - Node crash fully remedied immediately also

- **Con**
  
  - See other side of the ‘Pro’-coin ;) 
  - Play with the tradeoff using parameterization
GWiQ-P: Properties

- Small trees form around demand
  - Requests are remedied locally
- Coins are drawn towards ‘hot’ areas
  - Auto-Adaptable
- Fully distributed
  - No hot-spots & single points of failure
  - Low latency
  - No Congestion
  - Infinitely Scalable
- Fault Tolerant
Simulations’ (default) Properties

- **Toplogy** = BriteAS
  - Fast LANs, slower intercon/
- **NetSize** = 10K
- **Q/D**=1
  - Q for GWiQ; D for Overall demand
- **Change Rate** = 1%*D/E[EdgeDelay]
- **Demanders** = 1%*NetSize
- **Fail Rate** = 1%*Edges/E[EdgeDelay]
  - Applicable for FT scenarios
Simulations 1/6

- Topology: BriteAS
- Change rate: 1%*D/E[EdgeDelay]
- Q/D=1
- Demanders: 1%

GWiQ-P scalability due to locality
Simulations 2/6

- Topology: 10K BriteAS
- Demanders: 1%
Simulations 3/6

- Topology: 10K BriteAS
- $E[\text{EdgeDelay}] \sim 3$ ms
- Change rate: One Time $1\%*D$
- Demanders: 1%

Return to 99% sat after $\sim 30$ms
Simulations 4/6

- Topology: 10K BriteAS
- Change rate: 1%*D/E[EdgeDelay]
- Fail rate: 1%*Edges/E[EdgeDelay]
- Q/D=1
- Demanders: 1%

Plane depicts sat in ‘no-faults’ scenario

Threshold=0
Transactions

Excess coin exploitation
Simulations 5/6

- Topology: 10K BriteAS
- Change rate: 1%*$D/E[EdgeDelay]$
- Fail rate: 1%*$Edges/E[EdgeDelay]$
- Q/D=0.5
- Demanders: 1%

![Graph showing potential quota breach over time for different balancing factors.](chart)
Simulations 6/6

- Topology: BriteAS
- Change rate: $1\% \times D/E[\text{EdgeDelay}]$
- Fail rate: $1\% \times \text{Edges}/E[\text{EdgeDelay}]$
- $Q/D=1$
- Demanders: 1%

Same distribution, growing network. Locality in FT
Conclusion

- We displayed GWiQ-P, a Grid Wide Quota enforcement Protocol
- GWiQ-P is infinitely **scalable**
- GWiQ-P is fully **distributed**
- GWiQ-P is **local** hence very efficient
- GWiQ-P is **fault tolerant**
Thank You!

Q&A

Contact: Kfir Karmon
karmon@cs.technion.ac.il