Handling Collusion in Desktop Grids

Louis-Claude Canon, **Emmanuel Jeannot** and Jon Weissman

LORIA, CNRS, INRIA, Nancy
University of Minnesota

Scheduling in Aussois workshop

May18, 2008

Outline

Uncertainty

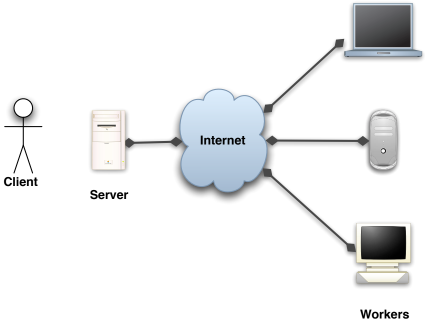
Uncertainty is everywhere

- Application level (submission date, duration, etc.)
- Environment level (volatility, failure, etc.)
- ⋆ User level (behavior, etc.)

Context

Desktop Grid

- A set of clients (submit jobs)
- A set of (volunteers) workers (execute jobs)
- A server (dispatch jobs, pull mode)
- Examples: Seti@home, BOINC, etc.



Context

Confidence in the workers

No control on the workers:

- Unreliable
- Malicious
- Victim of viruses

Kondo et al. 2007

- 35% of workers gives at least one wrong result
- 10% of workers commit 70% of errors
- Error rates over time vary greatly
- Error rates between two hosts often seem uncorrelated

Motivation

Desktop grid in presence of organized saboteurs

- For project with a small number of participants
- For project which are in "childhood"
- For security sake (military, medical, commercial)
- Against library or software bug on specific platform
- Against virus propagation
- Against Sybil attack (using pseudonymous identities)

Related work

Without collusion

- Job duplication or job verification
- Reputation system
- Majority vote
- 2 identical results are sufficient if no collusion and result space sufficiently large

With collusion

- Collusion = several workers send the same bad results
- [SASDA08]: "Secret" algorithm (unknown by the workers), based on reputation and majority voting + postpone decision (after all computations)

Outline

Computing model

Desktop grid

- n workers $w \in W$, one server and some jobs $j \in J$ (having a deadline)
- **Pull-based scheduling**: workers ask jobs to the servers
- **No volatility of resources** and machine supposed to be fully available (dedicated)

Error model

Different type of error

- Isolated I/O corruption
- Propagation of corrupted code/virus (malicious or not)
- Cheating, sabotage (organized or not)

Who : isolated workers only $\supset$ distinct groups $\supset$ overlapping groups

When : probability to give a wrong answer when several members of the group have the same job

Dynamicity : over time, colluding groups can change:

1. their probability of giving the wrong answer
2. their composition

Behavior model

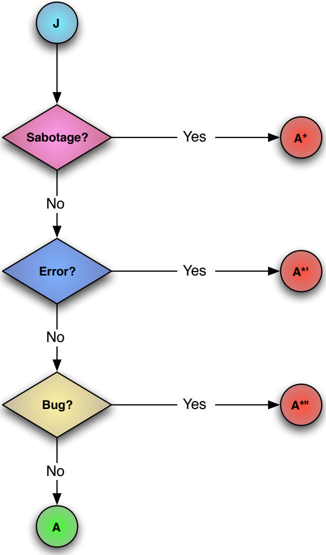
Behavior

- Individual error (no consistency)
- Buggy behavior (consistency)
- Sabotage (consistency and synchronization)

Model

- $\mathcal{P}(W)$: set of group of workers that gives the same wrong answer (buggy or sabotage). $k = |\mathcal{P}(W)|$
- P_r: probability to give an erroneous answer (reliability)
- $P_B(i)$: probability to have bug $i \in [1, k]$
- $P_s(i)$: probability of sabotage of group $i \in [1, k]$
- Some workers belongs to distinct buggy, sabotage groups
- Sabotage if at least two colluders are able to synchronize during the execution of the same job

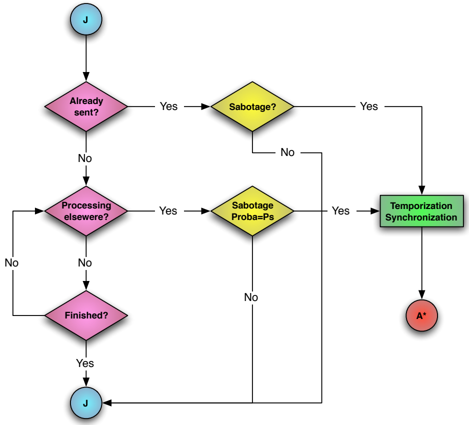
Worker algorithm



Sabotage behavior

Sabotage

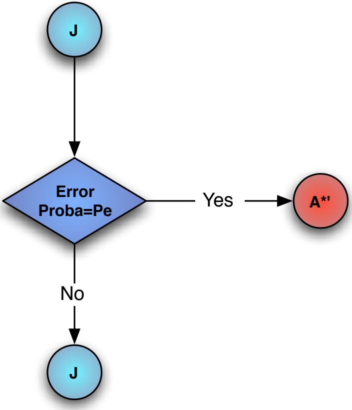
- Workers decide to sabotage or not a job depending on how numerous they are
- All workers in a sabotage group a have the same behavior (sabotage or not)
- The same answer is sent by workers in the same group
- If no sabotage the job is passed to the next step (Error)



Error behavior

I/O error, network error, etc

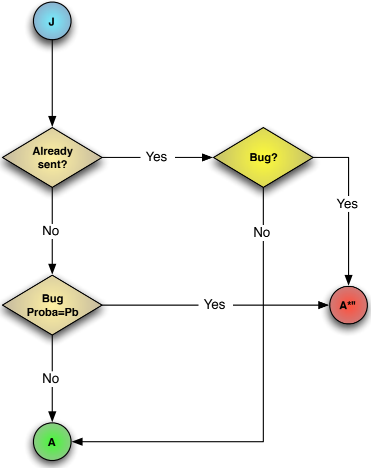
- The same job can be erroneous on some server and correct on some other
- Worker-dependant only behavior
- For the same job, different erroneous answers are sent
- $P_e = \frac{P_r}{1-P_s}$
- If no error the job is passed to the next step (bug)



Buggy behavior

Virus, buggy library, etc

- For a given job, worker in the same buggy group have the same behavior
- No synchronization necessary
- The same answer is sent by workers in the same group
- $P_b = \frac{1-\Pi_{i\in[1,k]}(1-P_B(i))}{1-P_s-P_e}$
- If no bug, the correct answer is returned



Problem description

Design server scheduling policy

- Dynamic job arrival
- Dispatch jobs onto workers (duplication allowed)
- Select the answer among the (possible) different ones
- Objectives:
  1. Maximize the number of correct answers
  2. Maximize throughput
- Issues:
  - Detect worker behavior
  - Outperform non collusion-aware strategies
  - Manage non-stationarity

Outline

Server policy framework

Framework

- Each job is duplicated to several workers
- For a given job j, a set of workers S_j is created and $\forall w \in S_j$ an answer a_w is generated
- Update worker reputation when receiving a new answer
- $\forall j \in J, S_j = S_j^1 \cup \ldots \cup S_j^k$ such that every worker of a given subset gives the same answer
- When all answers are received, select the one to return to the client

Worker group creation

Greedy/graceful mode

Graceful (over-duplication in order to acquire knowledge):

1. Find the smallest $k / \sum_{i=0}^{\lfloor \frac{k-1}{2} \rfloor} \binom{k}{i}(\alpha^i(1-\alpha)^{k-i}) < \epsilon$: number of workers such that the probability of bad workers does not form a majority (α upper-bound of bad worker fraction).
2. Assign the job to k workers chosen randomly

Greedy (lower duplication for improved throughput):

1. Select workers such that collusion likelihood is minimized (given by **reputation system**)
2. Add workers until the **answer selector** is confident in selecting the returned answer.

Graceful/greedy switching made by reputation system based on the certainty on the environment

Reputation system

Functional behavior

- Input: each answer sent by workers+the one chosen by the **answer selector**
- Output:
  - Fraction of colluders (saboteurs+buggy)
  - Collusion likelihood of a set of workers
  - Reliability of a given worker
  - Estimation of the confidence in the prediction

Data structures

- Collusion matrix or agreement matrix
- Reliability vector

Reputation system

Algorithms

- Determine the probability of colluding based on the subsets that have the same answer and the *chosen answer*
- Update the reliability vector for workers alone in a group
- Old value have a weight that decreases geometrically with time:

$$V_{n+1} = (1 - \alpha)V_n + \alpha X_n$$

(V: estimator and X : observation)

Answer selector

Collusion likelihood

$\forall j \in J, S_j = S_j^1 \cup \ldots \cup S_j^k$ such that every worker of a given subset gives the same answer

$$R_j^m = \Pr[\text{no collusion in } S_j^m] \times \prod_{n \neq m \wedge |S_n| \neq 1} \Pr[\text{collusion in } S_j^n]$$

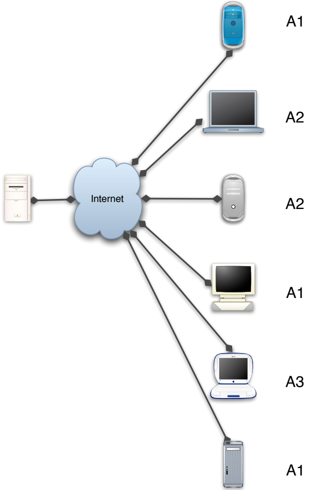
R_j^m: likelihood that answer m is the correct answer for job j

Return answer i where $i = \operatorname{argmax}(R_j^m)$, or $i = \operatorname{argmax}(|S_j^i|)$

Example

Answer selection

- 3 sub-group: S^1={1,4,6}, S^2={2,3} and S^3={5}
- $A1$ is selected as the good answer (majority)
- Entry (2,3) of collusion matrix is updated: $V_{2,3} = (1-\alpha)V_{2,3} + \alpha$
- Entry (1,4) (1,6) (4,6) of collusion matrix is updated: $V_{1,4} = (1-\alpha)V_{2,3}$
- Entry (1,2) (1,3) (4,2),. . . (6,3) of collusion matrix is updated: $V_{1,2} = (1-\alpha)V_{2,3}$
- Reliability vector is updated: (5) is decremented and the other are incremented



α value

$$V_{n+1} = (1-\alpha)V_n + \alpha X_n$$

Two cases

The value of α influence the importance of old measures.

We are able to compute the value of α in function of the standard deviation of the estimator:

1. long-term estimator $\alpha = 0.03$
2. short-term estimator $\alpha = 0.1$

Non-stationarity is detected if the two estimators are inconsistent $\Rightarrow$ inconsistancy $\in [0, 1]$: probability to use **graceful** policy (over-duplication)

Outline

Metrics

Servers objectives[1]

Accuracy or error rate #correct/#accepted

Overhead #replica/#accepted

Time time needed for executing all jobs (s)

Latency mean time for each job to be process (s)

Current jobs quantity of currently treated jobs

Throughput number of completed jobs / time-unit (FLOPS)

Effective throughput number of correct jobs / time-unit (FLOPS)

Idleness *fraction of unused CPU time*

[1] *Emphased metrics* are not yet implemented

Metrics, cont.

Algorithm objectives

Reliability vector accuracy norm of the difference between the estimated vector and the true one normalized to the number of known values (similar to the standard deviation with regards to the correct value)

Collusion matrix accuracy norm of the difference between the estimated matrix and the true one normalized to the number of known values (similar to the standard deviation with regards to the correct value)

Colluders fraction accuracy absolute difference between the estimated fraction and the real one

Outline

Experimental settings

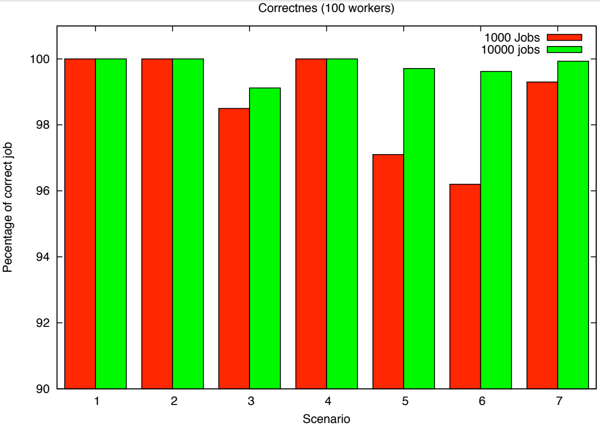
Settings

- Simulation with SimGRID (Java bindings)
- Test if the system is able to detect workers behavior

100 Workers, 1server, different scenarios

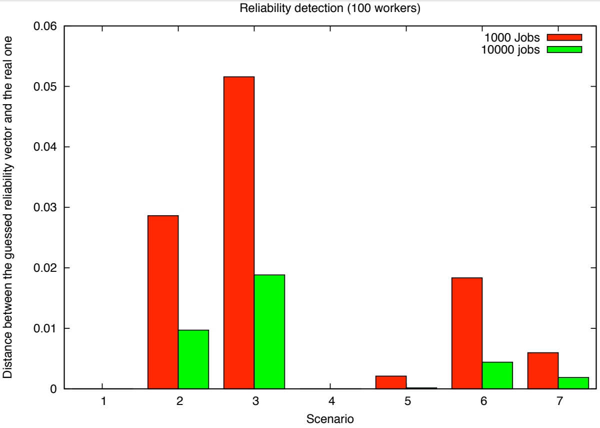
1. Workers are perfects
2. Some workers are really bad, majority is really good
3. 0.6 probability of failure (majority is harder to achieve than in 2)
4. Workers are perfectly reliable with one collusion group (30% of colluders always colluding)
5. Workers are perfectly reliable with one collusion group (30% of colluders and 30% of collusion)
6. 2 + 5
7. 2 + two collusion groups (40% of colluders and 30% of collusion)

Correctness



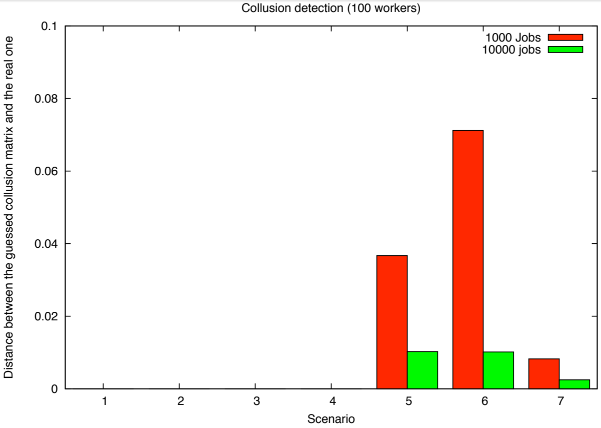
1. Workers are perfects
2. Some workers are really bad, majority is really good
3. 0.6 probability of failure (majority is harder to achieve than in 2)
4. Workers are perfectly reliable with one collusion group (30% of colluders always colluding)
5. Workers are perfectly reliable with one collusion group (30% of colluders and 30% of collusion)
6. 2 + 5
7. 2 + two collusion groups (40% of colluders and 30% of collusion)

Reliability detection



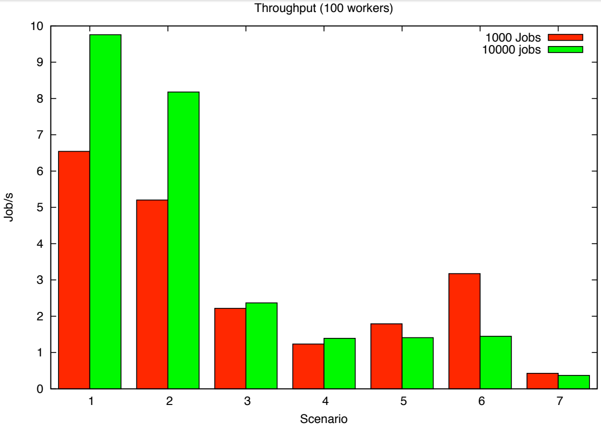
1. Workers are perfects
2. Some workers are really bad, majority is really good
3. 0.6 probability of failure (majority is harder to achieve than in 2)
4. Workers are perfectly reliable with one collusion group (30% of colluders always colluding)
5. Workers are perfectly reliable with one collusion group (30% of colluders and 30% of collusion)
6. 2 + 5
7. 2 + two collusion groups (40% of colluders and 30% of collusion)

Collusion detection



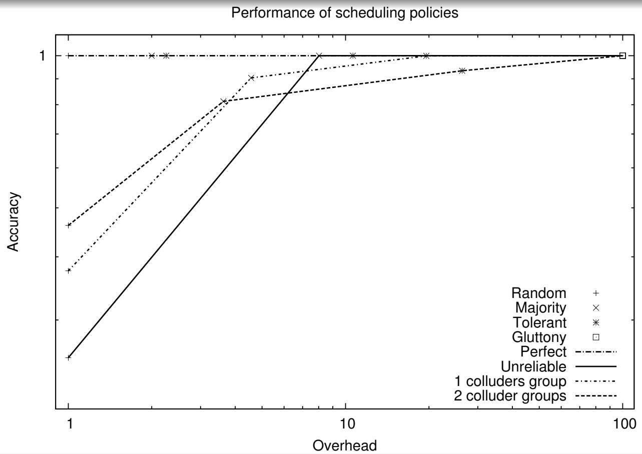
1. Workers are perfects
2. Some workers are really bad, majority is really good
3. 0.6 probability of failure (majority is harder to achieve than in 2)
4. Workers are perfectly reliable with one collusion group (30% of colluders always colluding)
5. Workers are perfectly reliable with one collusion group (30% of colluders and 30% of collusion)
6. 2 + 5
7. 2 + two collusion groups (40% of colluders and 30% of collusion)

Throughput



1. Workers are perfects
2. Some workers are really bad, majority is really good
3. 0.6 probability of failure (majority is harder to achieve than in 2)
4. Workers are perfectly reliable with one collusion group (30% of colluders always colluding)
5. Workers are perfectly reliable with one collusion group (30% of colluders and 30% of collusion)
6. 2 + 5
7. 2 + two collusion groups (40% of colluders and 30% of collusion)

Comparison with other strategies (100 workers)



1. **Perfect** : Perfect workers
2. **Unreliable** : 25% of reliability
3. **1 colluder group** : 50% of reliability+ 1 buggy collusion group (25% workers et 75% of having a bug)
4. **2 colluder groups** : unrelaliable + 1 buggy collusion group (25% workers et 100% of having a bug) + 1 sabotage collusion group (25% workers et 75% of sabotage)

1. **Random** : 1 worker/job
2. **Majority** : incremental group creation (up to majority of 2)
3. **Tolerant** : greedy/graceful + answer selector + reputation system + collusion aware
4. **Gluttony** : use all workers + majority selection

Outline

Conclusion, future work

Conclusion

- Desktop grids rely on volunteer workers
- Some worker have byzantine behavior
- Study the case of collusion
- Provide mechanism:
  - Correctly detect worker behavior
  - Provide a good accuracy
  - Throughput is lowered in case of huge adversity

Future work

- Enhanced comparison with other strategies
- Study the dynamic of groups (ex: virus propagation)
- Improve memory cost
- More abstraction $\Rightarrow$ analytical performance bound?