Replication and Redundancy in BOINC

Arnaud Legrand

Joint work with B. Gaujal, N. Gast (Inria Grenoble),
R. Righter, D. Anderson (UC Berkeley), W. Wu (CAS)

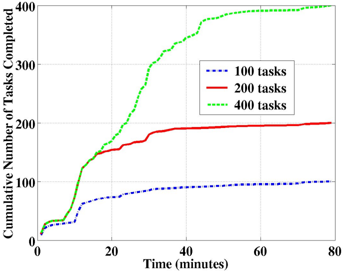
BOINC workshop, Budapest, September 2014

Outline

1. Improving BOINC Turnaround Time (Job Replication)
2. BOINC As a Storage Facility (Data Redundancy)

The Straggler Issue

- FCFS scheduling on a desktop Grid[KTB+04]



A.k.a the last finishing task issue

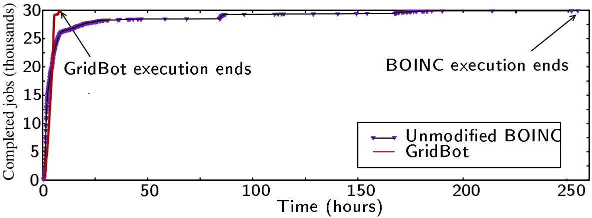
In BOINC, large deadlines and connection interval can make it worse.

- Can be quite problematic
  - Batch information and the corresponding files need to stay on the server (WCG) $\rightsquigarrow$ server overload
  - The system may starve when there is a limit on the number of active batches (CAS@home).
- Many solutions in the literature... but few implemented in practice
  - Exclude resources
  - Prioritize resources
  - Replicate jobs

The GridBot project

GridBot[SSGS09] (Technion - Israel Institute of Technology)

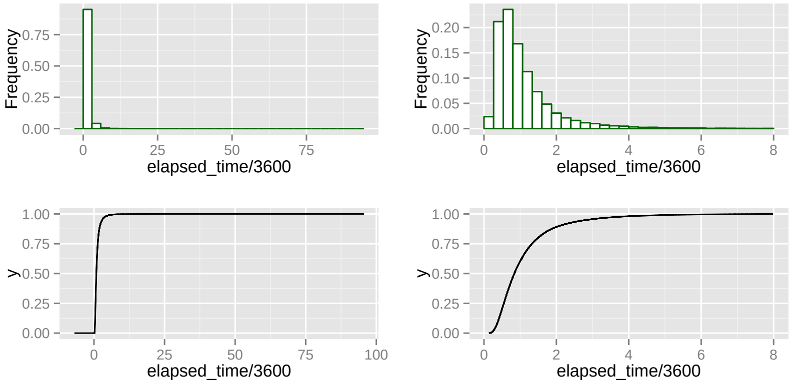
- Focus on response time of BoTs
- Use both community resources (BOINC) and grid resources (Condor)
- Better than BOINC and than Condor for this kind of workload
  - Replicate on reliable resources *toward the end*
  - Tighter deadlines for reliable resources (although you have to be careful with this. . .)



- Two other articles where BOINC is helped with reliable cloud resources
- Focus on the response time optimization of a single large batch

A Glance At The CAS@home Workload

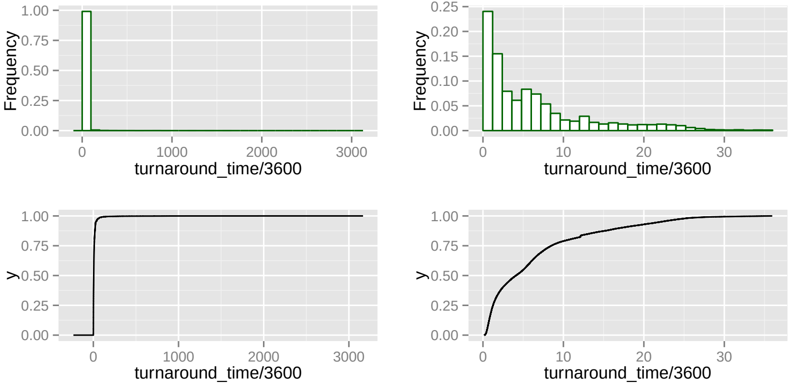
- Batches comprise 32 jobs with roughly the same computation workload.
- The running time of a job is .5 to 4 hours.
  - Jobs are short $\rightsquigarrow$ `elapsed_time` is not so different from `cpu_time`.
- Deadline is set to 36 hours



90% of the jobs take less than two hours to run

A Glance At The CAS@home Workload

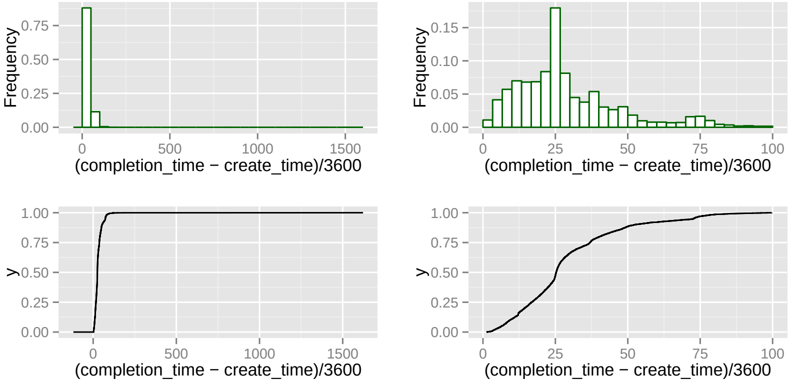
- Batches comprise 32 jobs with roughly the same computation workload.
- The running time of a job is .5 to 4 hours.
  - Jobs are short $\rightsquigarrow$ `elapsed_time` is not so different from `cpu_time`.
- Deadline is set to 36 hours



The job turnaround can be huge! (up to 5 months!)

A Glance At The CAS@home Workload

- Batches comprise 32 jobs with roughly the same computation workload.
- The running time of a job is .5 to 4 hours.
    - Jobs are short $\rightsquigarrow$ `elapsed_time` is not so different from `cpu_time`.
- Deadline is set to 36 hours

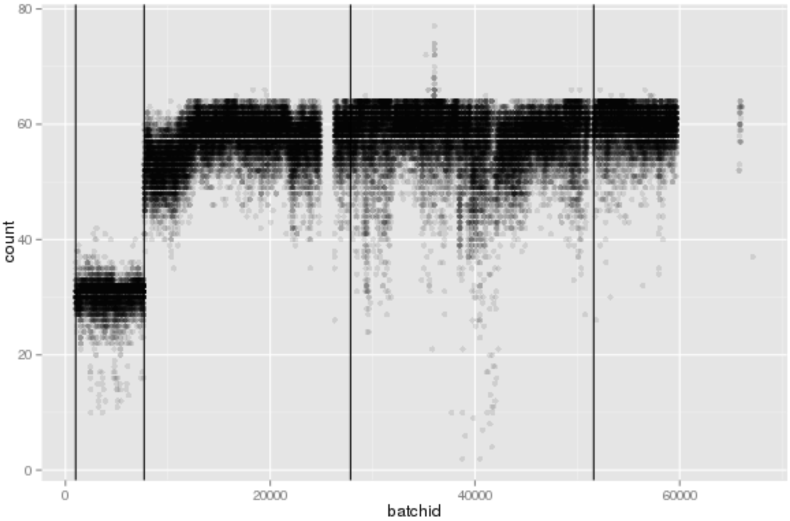


and so is the batch turnaround...

A Glance At The CAS@home Workload

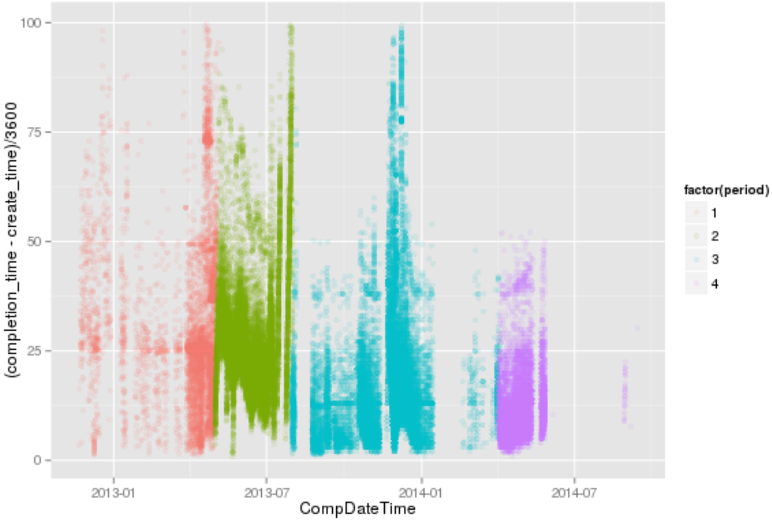
- Batches comprise 32 jobs with roughly the same computation workload.
- The running time of a job is .5 to 4 hours.
  - Jobs are short $\rightsquigarrow$ `elapsed_time` is not so different from `cpu_time`.
- Deadline is set to 36 hours
- CAS@home now has no more than 300 active batches at a time (a new batch comes in only when another one is completed) so the system can starve.
- At the moment: one additional replica for each job to improve the batch response time, which improves the system throughput despite the waste.

CAS@home: An Evolving System



Evolution of the number of jobs sent per batch

CAS@home: An Evolving System



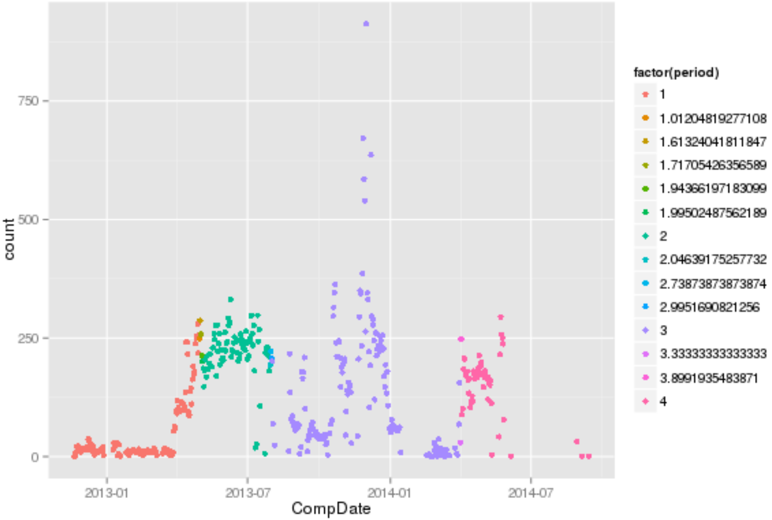
Evolution of the batch response time (hours)

CAS@home: An Evolving System

period	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1	1.25	16.61	25.24	29.46	37.65	99.75
2	0.98	19.15	24.96	28.04	32.90	99.45
3	1.00	9.00	14.01	18.98	23.01	99.49
4	1.00	7.00	11.01	13.05	16.01	52.01

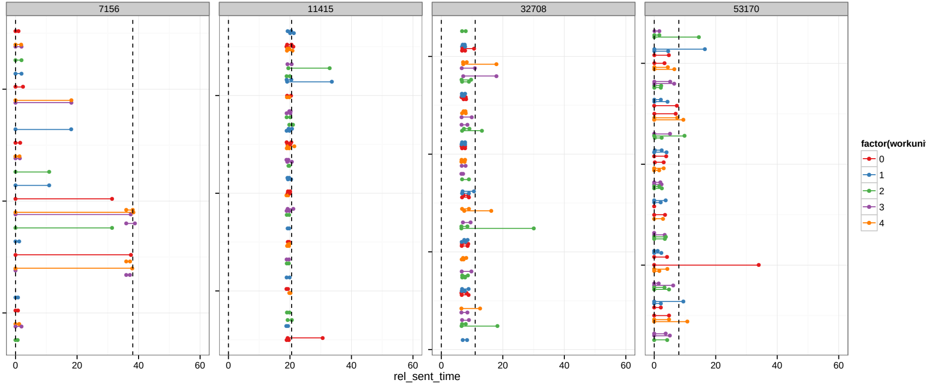
Evolution of the batch response time (hours)

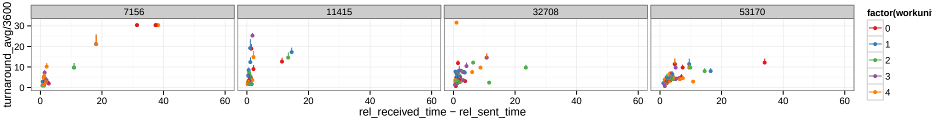
CAS@home: An Evolving System



Evolution of the batch throughput (batch per day)

CAS@home: An Evolving System





Evolution of the scheduling

What Can We Do About It?

- Improving job response time would help:
  - ~~Decrease the deadlines~~ (volunteers will complain)
  - Implement an "execute as possible" option
  - Implement a "report as possible" option
- The server can make a smarter use of resources. Whenever a host requests work, look for the right batch:
  - There is a continuum of behaviors and setting thresholds is difficult
  - Intuitively: use the fastest hosts to get rid of "almost finished" batches
  - We may want to "sacrifice" a batch to slow unreliable hosts so that they can still contribute without hurting response time
- Last week, we have crafted a simulation of BOINC and fed it with a profile of the CAS@home volunteers
- Short term work: check the modeling, test scheduling alternatives
- Long term work: handle non identical batches (SRPT), fair sharing between umbrella projects

Outline

1. Improving BOINC Turnaround Time (Job Replication)
2. BOINC As a Storage Facility (Data Redundancy)

BOINC As a Storage Facility

Volunteer computing is based on the idea that idle personal computers could as well be used to make distributed computations.

David coined a few years ago that we could do the same with storage space.

Disk space average 50 GB available per client $\leadsto$ 35 Petabytes total

Trends disk sizes increasing exponentially, even faster than processing power.

- 1 TB $\times$ 1M clients = 1 Exabyte

Could we construct a distributed "data center" from empty disk space from volunteers?

Volunteer Data Archival

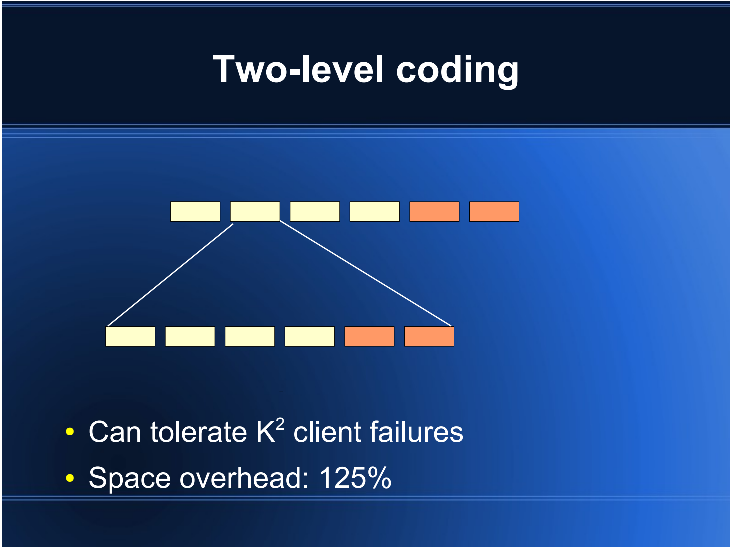
Same difficulty as usual:

- Volunteers are unreliable resources. They may leave (or enter) the system at any time, destroying whatever data and computations thy have been storing.
- Volunteers cannot be easily contacted. In BOINC, we need to volunteers to contact the server.

Our goal is to design a reliable data storage out of unreliable volunteers by coding data and storing redundant chunks in volunteers.

- Files originate on server
- Chunks of files are stored on clients
- Files can be reconstructed on server (with high latency)
- Design Goals
    - arbitrarily high reliability (99.999%)
    - support large files

David's Proposal : Two-Level Coding + Replication



Open questions: Why two levels ? How much redundancy ?

Using a Cost Model...

Assumptions:

- Single file split in N chunks
- Local storage is expensive: holding cost of H per time unit and per chunk.
- Each volunteer stores one chunk of data
- Erasure coding: the whole file is encoded with $M \geq N$ chunks but any N chunks out of M can be used to recreate the file
  - The server can create and upload a chunk to a volunteer iff it has N chunks in its own memory.

How to choose the best redundancy $M - N$?

. . . and Markov Chains

Volunteers are independent from each others so we can model most events as Poisson process.

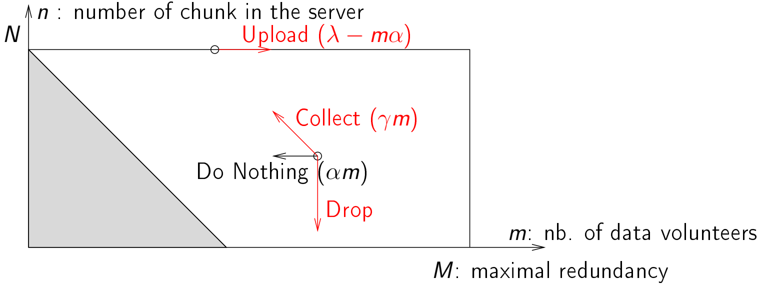
- New volunteers join the system and request data with rate λ.
- Each volunteer that is already storing a chunk is called a data volunteer.
  - Data volunteers contact the server at rate γ (in which case the server can download its chunk)
  - Data volunteers leave the system at rate α (in which case its chunk is lost)

At any time, the state of the system is characterized by (n, m):

- n is the number of chunks stored locally
- m is the number of data volunteers

If the file is lost (when the system moves to (n, m) where $N < n + m$) a large cost C is incurred and we go to $(N, 0)$.

Control Actions



The server has four available actions:

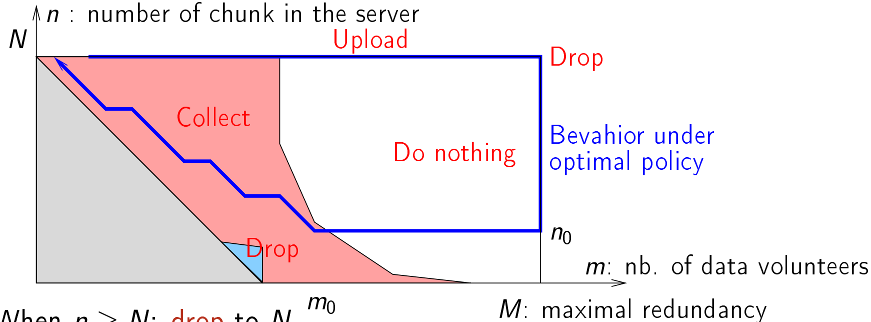
Upload changes new volunteers into data volunteers. As long as $n \geq N$, this is possible $\boxed{\text{whenever a new volunteer arrives}}$. The state changes to $(n, m+1)$.

Collect $\boxed{\text{Whenever a data volunteer arrives}}$, the server can collect its chunk. The state changes to $(n+1, m-1)$.

Drop erases any $k \leq n$ chunks from memory, changing the state from (n, m) to $(n-k, m)$.

Do Nothing in which case some chunks are lost when data volunteers leave

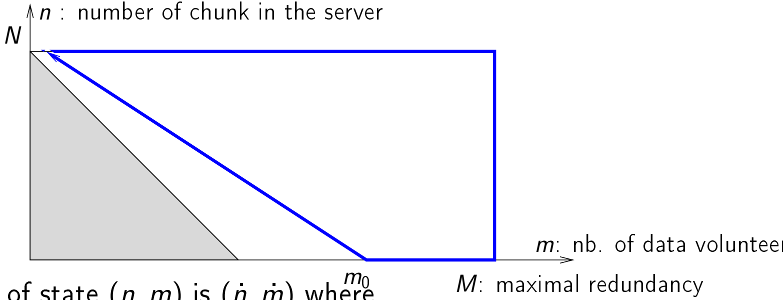
What Does The Optimal Policy Look like ?



- When $n \geq N$: drop to N
- When we have the whole file ($n = N$), as long as $m < M$, there is nothing to loose in uploading the file (except the holding cost).
- When we reach state (N, M), it will be optimal to immediately drop $N - n_0 > 0$ chunks for some n_0.
- There are two switching curves $f_1(m) \geq f_2(m)$, such that:
  - for $n \geq f_1(m)$ it will be optimal to do nothing,
  - for $f_2(m) \leq n < f_1(m)$ it will be optimal to collect chunks,
  - for $n < f_2(m)$ it will be optimal to drop chunks.

Fluid Approximation

Computing f_1 and f_2 for a given N and M is very hard. However, when N and M go to infinity, things average out (fluid approximation).

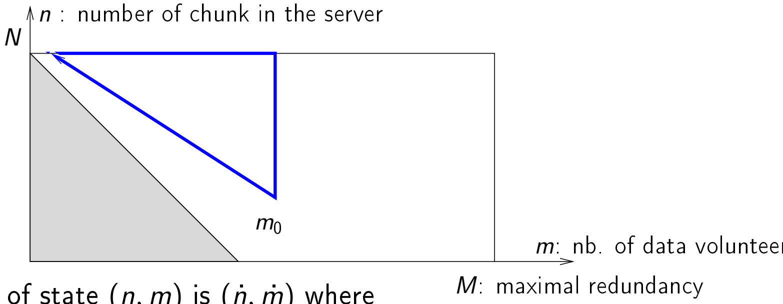


The rate out of state (n, m) is $(\dot{n}, \dot{m})$ where

- $(\dot{n}, \dot{m}) = (0, -m\alpha)$ if the action is to do nothing,
- $(\dot{n}, \dot{m}) = (0, \lambda - m\alpha)$ if $n = N$ and the action is to upload,
- $(\dot{n}, \dot{m}) = (m\gamma, -m(\gamma + \alpha))$ if the action is to collect,
- and the fluid immediately drops from (n, m) to (n_0, m) if the action is to drop $n - n_0 \geq 0$ of " chunk fluid".

Fluid Approximation

Computing f_1 and f_2 for a given N and M is very hard. However, when N and M go to infinity, things average out (fluid approximation).

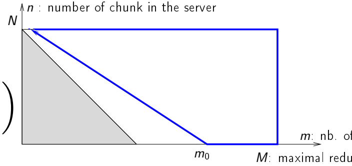


The rate out of state (n, m) is $(\dot{n}, \dot{m})$ where

- $(\dot{n}, \dot{m}) = (0, -m\alpha)$ if the action is to do nothing,
- $(\dot{n}, \dot{m}) = (0, \lambda - m\alpha)$ if $n = N$ and the action is to upload,
- $(\dot{n}, \dot{m}) = (m\gamma, -m(\gamma + \alpha))$ if the action is to collect,
- and the fluid immediately drops from (n, m) to (n_0, m) if the action is to drop $n - n_0 \geq 0$ of " chunk fluid".

Cost for the Fluid Approximation

1. Starting from $(N, 0)$ we upload and move to (N, M) at rate $(\dot{n}, \dot{m}) = (0, \lambda - m\alpha)$.



$$\begin{cases} t_1 & = -\frac{1}{\alpha}\ln\left(1 - \frac{\alpha}{\lambda}M\right) = \frac{1}{\alpha}\ln\left(\frac{\lambda}{\lambda - \alpha M}\right) \\ C_1 & = HNt_1 = \frac{HN}{\alpha}\ln\left(\frac{\lambda}{\lambda-\alpha M}\right) \end{cases}$$

2. We immediately drop to state $(0, M)$ at t_1 at no cost.
3. From $(0, M)$, do nothing and move to $(0, m_0)$ at rate $(0, -m\alpha)$.

$$t_2 = \left(\ln(M) - \ln(N(\alpha+\gamma)/\gamma + 1)\right)/\alpha \text{ and } C_2 = 0$$

4. From $(0, m_0)$ we collect new chunks and move back towards $(N, 1)$ at rate $(\dot{n}, \dot{m}) = (\gamma m, -(\gamma+\alpha)m)$.

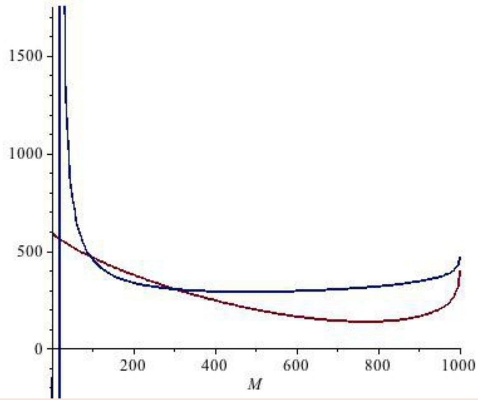
$$\begin{cases} t_3 & = \frac{1}{\gamma+\alpha}\ln(N(\alpha+\gamma)/\gamma+1) \\ C_3 & = \frac{H\gamma}{(\gamma+\alpha)^2}(1 + (N(\alpha+\gamma)/\gamma+1)\ln(N(\alpha+\gamma)/\gamma+1)) \end{cases}$$

In both cases, the total average cost is $\boxed{V = \dfrac{C_1 + C_3}{t_1 + t_2 + t_3}}$

Optimal Redundancy

We get $V = H \dfrac{N \ln\left(\frac{\lambda}{\lambda - \alpha M}\right) + \frac{\gamma\alpha}{(\gamma+\alpha)^2}(1 + m_0 \ln(m_0))}{\left(\ln\left(\frac{\lambda}{\lambda-\alpha M}\right) + \ln M\right) - \frac{\gamma}{\gamma+\alpha}\ln(m_0)}$

With $\alpha = 1$, $\gamma = 1$, $\lambda = 300$, $N = 30$, $H = 20$, we get:



The two level coding is not in the picture yet but it seems feasible to incorporate it.

References

Derrrick Kondo, M. Taufer, C. Brooks, Henri Casanova, and Andrew A. Chien.
Characterizing and Evaluating Desktop Grids: An Empirical Study.
In *Proc. of the Intl. Parallel and Distributed Processing Symp. (IPDPS)*, 2004.

Mark Silberstein, Artyom Sharov, Dan Geiger, and Assaf Schuster.
Gridbot: execution of bags of tasks in multiple grids.
In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, New York, NY, USA, 2009. ACM.