SANDIA REPORT

SAND2019-5139
Unlimited Release
Printed May 2019

Sandia National Laboratories

Some Results About Distributed Load Balancing

Philippe P. Pébaÿ, Jonathan Lifflander

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology and Engineering Solutions of Sandia, LLC.

SAND2019-5139
Unlimited Release
Printed May 2019

Some Results About Distributed Load Balancing

Philippe P. Pébaÿ, Jonathan Lifflander
08753, 08753
Sandia National Laboratories
Livermore, CA 94551
U.S.A.

Abstract

In this report with discuss load-balancing research and results in the context of the DARMA/VT project.

This page intentionally left blank.

This document was generated on 2019-05-08 with the Automatic Report Generator (ARG) version "master" on a Darwin system.

This page intentionally left blank.

Contents

List of Figures

List of Tables

This page intentionally left blank.

Chapter 1

Introduction

The goal of this report is to discuss the findings we made with the Load-Balancing Simulator (LBS), and also to discuss some theoretical results.

This page intentionally left blank.

Chapter 2

Remarks on the Grapevine Load-Balancing Algorithm

2.1 Informed Transfer Criterion

Here we discuss the original *informed transfer* algorithm of the Grapevine algorithm. We begin with line 6 of said algorithm:

$$6: \quad \textbf{if } (L_X + load(O_i) < L_{avg}) \textbf{ then}$$

This is motivated by our observations with our first experiments with a strict Grapevine implementation of a high rejection rate of this condition. Furthermore, this rejection rate only slowly and marginally decreased with the increase in the number of *gossiping rounds* (k) in the first phase (gossip algorithm) of Grapevine.

2.1.1 Convergence Problems

In fact, we noticed that this finding can be explained by the fact that the condition set forth in the algorithm for each overloaded processor:

$$4: \quad \textbf{while } (L_i > (T \times L_{ave}))$$

mandates that, after a variable number of iterations, each overloaded processor no longer is, up to a certain relative threshold T. This implies that, in the worst case, after completion of this loop,

$$L_{\max} \leq T \times L_{ave} \quad \Longleftrightarrow \quad \mathcal{I}_{\mathcal{D}} < T - 1$$

where $\mathcal{I}_D$ is the load imbalance of the distribution D of objects across the entire set of processors. This amounts to saying that the objective function that the algorithm aims to minimize is $F(D) = \mathcal{I}_{\mathcal{D}} - T + 1$, and that a sufficient stopping criterion if $F(D) \geq 0$ (we observe that L_{ave} is by definition a constant as no loss or gain of load may occur globally).

However, $F(D) \geq 0$ is by no means necessary, and it fact if it were, there would be no guarantee that the algorithm would terminate in finite time. This is however ensured by the

fact that, given an underloaded processor, the criterion of line 6 is tested for all of its objects, of which there is only a finite number. While this ensures termination of the while-loop in finite time – this does not guarantee that *any* transfer will have occurred at all.

We now illustrate this problem with one representative test case with $o = 10^4$ objects initially distributed across $p = 2^4$ amongst $n = 2^{12}$ processors, of $i = 10$ iterations of the original Grapevine algorithm (via our NodeGossiper simulator), each of each having $k = 10$ gossiping rounds, with an overload threshold of $T = 1.0$ and a fanout factor of $f = 6$. We observe the following rejection rates caused by the criterion of line 6:

iteration (index)	transfers (number)	rejected (number)	rejection rate (%)	imbalance ($\mathcal{I}$)
0				280
1	9084	154931	94.46	187
2	4	1654	99.76	187
3	1	1130	99.91	187
4	7	2682	99.74	185
5	6	2396	99.75	183
6	2	1143	99.83	183
7	1	1041	99.90	183
8	0	882	100.0	183
9	0	882	100.0	182
10	3	1405	99.79	182

These immediately hint at the fact that the decision criterion is too tight. And indeed, it enforces strict monotonicity for each of the underloaded processors; in other words, it uses the "taxicab" norm ($||\cdot||_1$) to minimize in the $||\cdot||_\infty$ sense: this criterion is therefore not adapted to the considered minimization problem. Another way to look at the problem is to see it as an attempt to decrease the load of underloaded processors while never allowing a single underloaded one to become overloaded – even when this may improve the global imbalance.

As a result, these almost full rejection rates, limits load-balancing to a noticeable decrease of $\mathcal{I}$ during the first iteration of the algorithm, after which esssentially no further improvements occur while $\mathcal{I}$ remains stuck in a local minimum. Furthermore, increasing k and allowing for higher values of T does not substantially affect the outcome on average (with the exception of the occasional convergence due to a nice original layout). This is illustrated in Figure 2.1.

2.1.2 An Optimal Decision Criterion

In order to ensure a faster convergence of the algorithm, we propose the following adapted criterion:

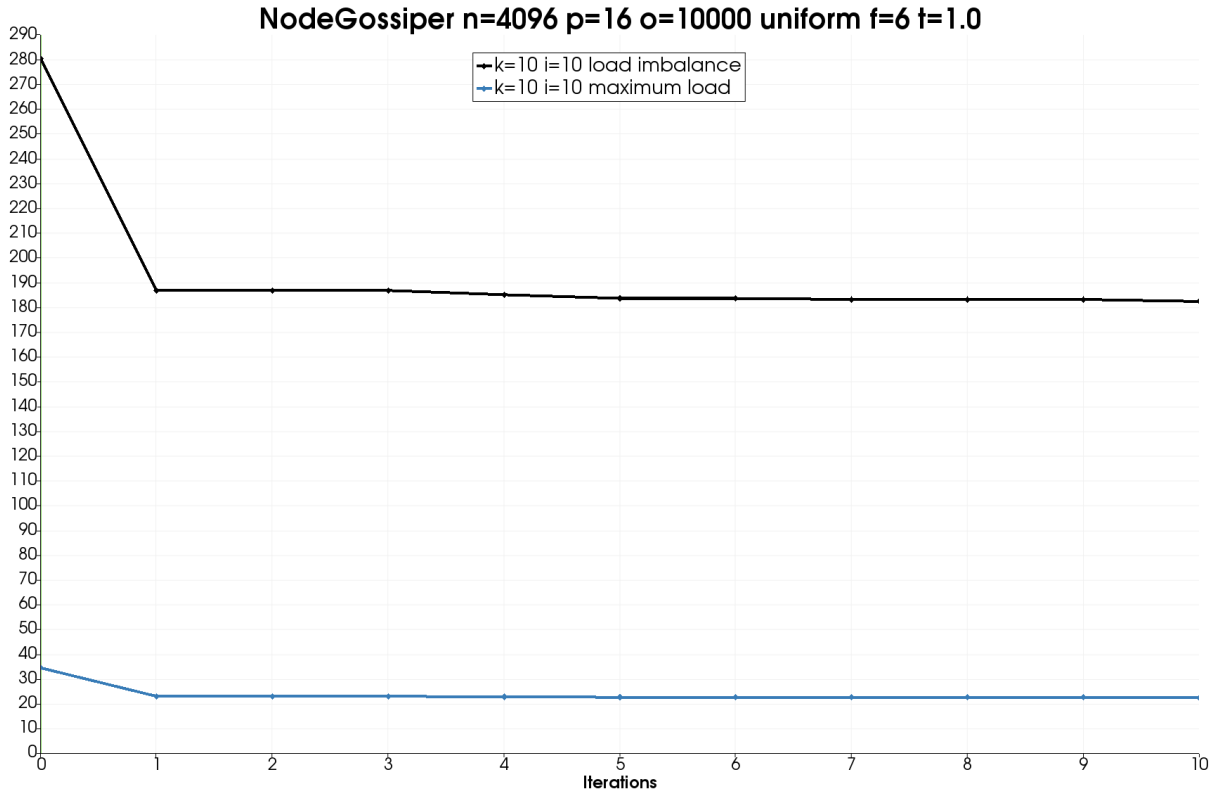

Figure 2.1: Plot of global imbalance ($\mathcal{I}$) and maximum load ($L_{\max}$) versus iterations.

Lemma 1:
The following alternate criterion:

$$6' : \qquad \textbf{if } load(O_i) < L_i - L_X \textbf{ then}$$

ensures that the objective function F monotonically decreases.

Proof:
Consider a strictly overloaded processor p_i and a strictly underloaded one p_X in an processor/object distribution D, with respective loads L_i and L_X. Consider also an object $O_i \in p_i$ such that $\ell = load(O_i) < L_i - L_X$ (both sides being necessarily positive by hypothesis on loads); we can thus distinguish the two following disjoint cases:

1. If $\ell \leq \frac{1}{2}(L_i - L_X)$, then $L_i - \ell \geq L_X + \ell$ and thus $L_X + \ell \leq L_i - \ell < L_i$.
2. If $\ell > \frac{1}{2}(L_i - L_X)$, then $L_i - \ell < L_X + \ell$ and thus $L_i - \ell < L_X + \ell < L_i \Leftrightarrow \ell < L_i - L_X$.

Therefore, we can assert that, overall, $\max(L_i - \ell, L_X + \ell) < L_i \Leftrightarrow \ell < L_i - L_X$. Now, recall that

$$F(D) = \frac{L_{\max}}{L_{ave}} - T \geq \frac{L_i}{L_{ave}} - T,$$

with equality if and only if p_i is such that $L_i = L_{\max}$ in D (i.e., it has maximum load in the original distribution). Therefore, $\ell < L_i - L_X$ ensures that

$$\frac{\max\left(L_i - \ell, L_X + \ell\right)}{L_{ave}} - T < \frac{L_i}{L_{ave}} - T \leq F(D) \tag{2.1}$$

Finally, when considering the new load/processor distribution D' that results from the transfer of O_i from p_i to p_X, and denoting $L'_i = L_i - \ell$ and $L'_X = L_X + \ell$ their respective new loads, we are faced with a trichotomy of possible cases (the two first ones are not necessarily disjoint):

1. If p_i is such that $L'_i = L'_{\max}$ in D' (i.e., p_i has maximum load in the new distribution D'), then $\max\left(L_i - \ell, L_X + \ell\right) = L_i - \ell = L'_i$ and (2.1) yields:

$$F(D') = \frac{L'_{\max}}{L_{ave}} - T = \frac{L'_i}{L_{ave}} - T = \frac{L_i - \ell}{L_{ave}} - T < \frac{L_i}{L_{ave}} - T \leq F(D)$$

2. If p_X is such that $L'_X = L'_{\max}$ in D' (i.e., p_X has maximum load in the new distribution D'), then $\max\left(L_i - \ell, L_X + \ell\right) = L_X + \ell = L'_X$ and (2.1) yields:

$$F(D') = \frac{L'_{\max}}{L_{ave}} - T = \frac{L'_X}{L_{ave}} - T = \frac{L_X + \ell}{L_{ave}} - T < \frac{L_i}{L_{ave}} - T \leq F(D)$$

3. If a processor $p_Y \notin \{p_i, p_X\}$ is such that $L'_Y = L'_{\max}$ in D' (i.e., neither p_i nor p_X have maximum load in the new distribution D'), then necessarily $L'_Y = L_Y$ because the transfer did not affect p_Y, and thus necessarily $L'_Y \leq L_i$ and (2.1) yields:

$$F(D') = \frac{L'_{\max}}{L_{ave}} - T = \frac{L'_Y}{L_{ave}} - T = \frac{L_Y}{L_{ave}} - T \leq \frac{L_i}{L_{ave}} - T \leq F(D).$$

   Furthermore, in this case, $\frac{L_Y}{L_{ave}} - T \leq \frac{L_i}{L_{ave}} - T$ is *not* strict if and only if P_Y also had maximum load in D; because there is only a finite number of such maximally-overloaded processors, we are guaranteed that in a finite number of iterations the inequality will become strict.

We can therefore conclude that, overall,

$$\ell < L_i - L_X \Longrightarrow F(D') < F(D).$$

□

We remark that the new, modified criterion can be equivalently written as

$$\mathsf{6}' : \qquad \textbf{if } L_X + load(O_i) < L_i \textbf{ then}$$

which is indeed less strict than the the original one, for it allows, in particular, one underloaded processor to land in overloaded territory after a transfer. However, what is ensured is

the maximum norm will not increase. In addition, Lemma 1 ensures that, as long as one can find at least one object satisfying this criterion on at least one overloaded processor, then the optimization can continue. However, once it is no longer possible to find such a combination, then F may no longer decrease: this new criterion thus also provides a stopping criterion.

While this criterion will provide more opportunities for overload transfers than the original one, one may wonder whether it could not be further relaxed, hereby allowing for even lower rejection rates. This question is quikcly answered by the following, with the same notations a in Lemma 1:

Lemma 2:
If p_i is a processor with maximum load in D, and

$$(\exists O_i \in p_i)\ (\exists p_X \in D) \quad \ell = load(O_i) \geq L_i - L_X$$

then if O_i is transferred from p_X to p_i, the objective function F does not decrease (and possibly increases).

Proof:
If p_i has maximum load in the object/load distribution D, and one can find $O_i \in p_i$ and $p_X \in D$, then by definition of F one has, on one hand:

$$\frac{L_X + \ell}{L_{ave}} - T \geq \frac{L_i}{L_{ave}} - T = F(D).$$

On the other hand, in the new distribution D' obtained by transferring O_i from p_i to p_X, one has:

$$\frac{L_X + \ell}{L_{ave}} - T = \frac{L'_X}{L_{ave}} \leq \frac{\max L'_\cdot}{L_{ave}} - T = F(D').$$

Combining the two above inequalities thus yields $F(D') \geq F(D)$. Furthermore, if (O_i, p_x) is such that L'_X is not a maximally-overloaded processor in the new distribution, then the latter inequality is strict, in which case F increases. □.
As a result of Lemma 1 and Lemma 2, we can now assert the following:

Proposition [Optimal Load-Transfer Criterion]:
The following alternate criterion:

$$6' : \qquad \textbf{if } load(O_i) < L_i - L_X \textbf{ then}$$

is optimal for the load transfer strategy of Algorithm 2.

Proof:
From Lemma 1 we know that this alternate criterion ensures that monotonicity is sufficient to ensure that F monotonically decreases.

Furthermore, from Lemma 2 we know that if this criterion is not met for at least one particular case, then F will no longer monotonically decrease (and will possibly increase if (O_i, p_X) is such that L'_X is not maximal in D'.

Therefore, alternate criterion `6'`, being necessary and sufficient, is optimal for the considered optimization strategy. □.

2.1.3 Results with Modified Criterion

We now study the results of the modified algorithm, with alternate criterion `6'`, when used in the NodeGossiper simulator for the same case as above. The new acceptance/rejection results are as follows:

iteration (index)	transfers (number)	rejected (number)	rejection rate (%)	imbalance ($\mathcal{I}$)
0				280
1	11292	648	5.427	3.34
2	4044	3603	47.12	1.60
3	2201	3412	60.79	0.873
4	1324	3586	73.03	0.632
5	765	3171	80.56	0.632
6	410	2969	87.87	0.626
7	247	2794	91.88	0.626
8	159	2749	94.53	0.626
9	120	2682	95.72	0.626
10	72	2643	97.35	0.623

In contrast with what was happening with the original criterion `6`, we see now that the rejection rate is almost null initially, then slowly increases as the global imbalance rapidly decreases. In fact, with already more than acceptable values of $\mathcal{I}$, additional iterations continue to improve the outcome, hereby experimentally validating the preceding theoretical results. This is examplified by comparing the values of $\mathcal{I}$ in both cases:

iteration (index)	criterion `6` ($\mathcal{I}$)	criterion `6'` ($\mathcal{I}$)
0	280	280
1	187	3.34
2	187	1.60
3	187	0.873
4	185	0.632
5	183	0.632
6	183	0.626
7	183	0.626
8	183	0.626
9	182	0.626
10	182	0.623

We note, in particular, that the modified algorithm has not fully run its course after iteration 10, and continues to improve $\mathcal{I}$, albeit modestly, while the original algorithm has essentially converged to a very sub-optimal local minimum and is no longer able to improve the overall imbalance after a few steps.

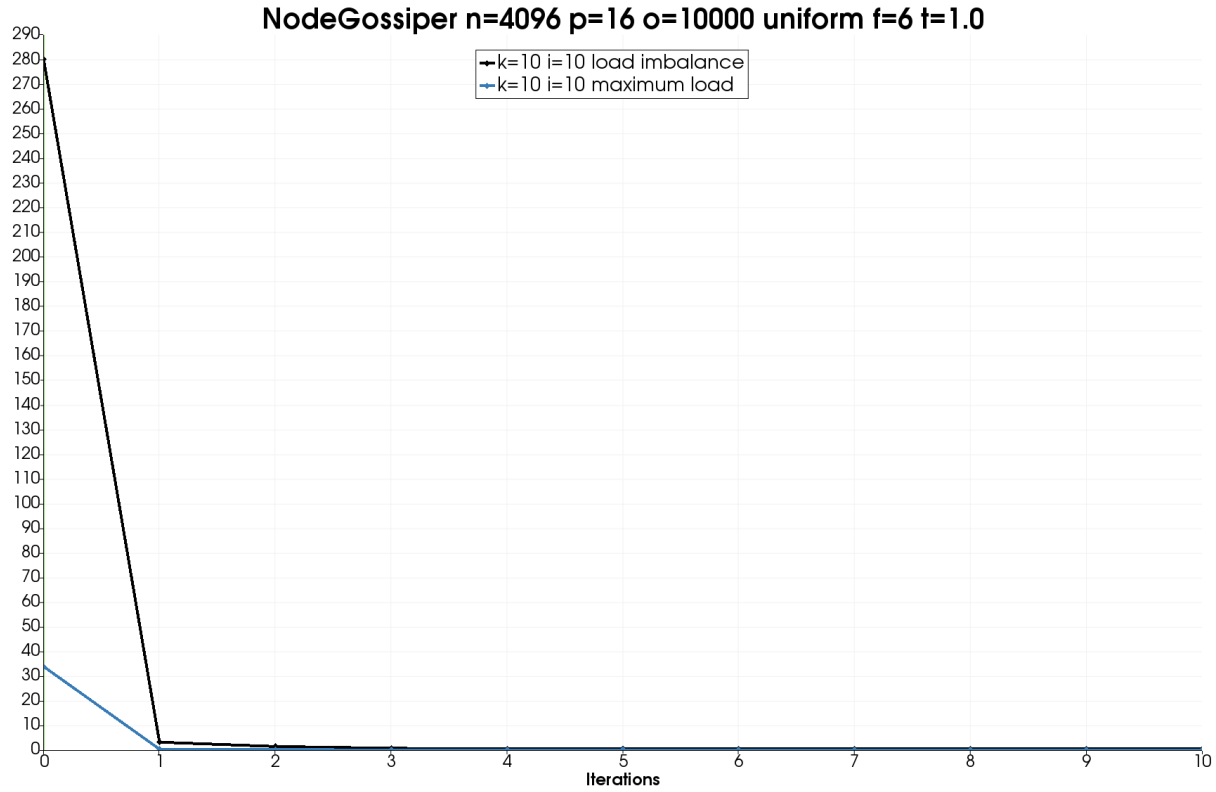

Figure 2.2: Plot of global imbalance ($\mathcal{I}$) and maximum load ($L_{\max}$) versus iterations with modified criterion `6'`.

Furthermore, we illustrate below that, *ceteris paribus* increasing the number of objects in order to obtain an object-to-processor ratio of 8 (closer to our typical use case than the ratio of about 2.5 used above) allows for even faster convergence:

iteration (index)	transfers (number)	rejected (number)	rejection rate (%)	imbalance ($\mathcal{I}$)
0				270
1	33761	269	0.790	1.52
2	7179	3642	33.66	0.714
3	4227	5946	58.45	0.399
4	3304	9023	73.20	0.317
5	2496	9570	79.31	0.273
6	1978	10138	83.67	0.187
7	1419	9895	87.46	0.139
8	1026	9312	90.08	0.139
9	692	8938	92.81	0.139
10	463	8379	94.76	0.139

These results are consistent with what the intuition would assume: having relatively more objects per processor provides more flexibility for the load-balancer to exploit. This is even further illustrated by the fact that, despite having attained an even better-balanced distribution than before (0.139 vs. 0.623), the algorithm continues to progress with even lower terminal rejection rates than before.

2.2 Performance Model

To further elucidate the theoretical tradeoff between utilizing more iterations (i) or gossiping rounds (k), we build a performance model for the load balancing algorithm to model the computational complexity.

The implementation of the Grapevine algorithm relies on having efficient termination detection, an extensively well-researched distributed computing problem. A distributed computation is globally terminated if every process is locally terminated and there is no message in transit between any processes. Termination detection is an important distributed problem, on which many non-bulk-synchronous algorithms rely in order to correctly sequence operations across complex dynamic communication patterns. In Grapevine, it is used to determine when gossiping is finished and when all load transfers have occurred.

Chandy and Misra [1] prove the lower and upper bounds on the control message complexity T_m for any termination detection algorithm, where $|\mathbf{M}|$ is the total number of messages in an underlying computation and $|\mathbf{P}|$ is the number of processors.

$$T_m = O(|\mathbf{M}|) = \Omega(|\mathbf{P}|).$$

Intuitively, the worst-case bound occurs when for every computation message, the termination detection algorithm is activated. The "dynamic" Dijkstra-Scholten termination algorithm [2] for diffusing computations utilizes an engagement tree to obtain this bound tightly

but never performs better—even in "best" case situations— than $\Theta(|\mathbf{M}|)$. For distributed applications in less tightly coupled applications or cases with partial processor engagement in the computation, this algorithm tends to perform well in practice. Thus, even the best case time complexity T_c for the Dijkstra-Scholten algorithms is very costly (and is notably not scalable with respect to $|\mathbf{P}|$):

$$T_c = \Omega(|\mathbf{P}|).$$

In contrast, in tightly-coupled cases when all processors are engaged in the computation, as in the Grapevine algorithm, $O(|\mathbf{M}|)$ is overly costly to assume in practice. Thus, the 4-counter, wave-based termination algorithm is often applied in these contexts that reaches a message complexity lower bound of $2 \times |\mathbf{P}|$ in the best case scenario, but costs $O(|\mathbf{M}| \times |\mathbf{P}|)$ in the worst case. However, without arbitrary delays in the computation across all the processors, a small factor on top of $2 \times |\mathbf{P}|$ is a highly probable outcome when utilizing termination to sequence phases in Grapevine. With the message complexity, we can also bound the time complexity for the 4-counter termination detection algorithm:

$$T_c = O(|\mathbf{M}| \times \log(|\mathbf{P}|)) = \Omega(2 \times \log(|\mathbf{P}|)).$$

With T_c in place, we bound the time complexity of the Grapevine algorithm, for a single iteration, denoted as G_c. We assume that f, the fanout during the gossip phase, is a small constant, where $f \ll P$.

$$G_c = O(f \times \min(k, \log_f(|\mathbf{P}|)) + |\mathbf{O}| + 2 \times T_c)$$

where:

$$\begin{aligned} k &= \text{the number of gossiping rounds} \\ f &= \text{the fanout during gossip where} f \ll |\mathbf{P}| \\ \mathbf{P} &= \text{the set of processors} \\ \mathbf{O} &= \text{the set of objects where } |\mathbf{O}| = O(|\mathbf{P}|). \end{aligned}$$

The Grapevine algorithm, as originally described, can cost $|\mathbf{O}|$ in the worst case during the transfer phase. This is observed when the load L_i of a single processor is composed of all objects $\mathbf{O}$ in the system. The original algorithm contains a while loop that will attempt to reassign all these objects living a single processor during a single iteration of the Grapevine algorithm. For the Grapevine algorithm to be scalable in the worst case, we must bound the number of possible transfers at each iteration by $\log(|\mathbf{P}|)$.

2.3 Rounds vs. Iterations

We now empirically examine the comparative effect in terms of load imbalance as well as of maximum load, of simultaneously increasing and decreasing the number of iterations and rounds, so that $i \times k$ remains constant.

This is illustrated here in the case of $o = 10^4$ objects initially distributed across $p = 2^4$ amongst $n = 2^{12}$ processors, of the NodeGossiper simulator with an overload threshold of $T = 1.0$ and a fanout factor of $f = 6$. The object times were also pseudo-randomly generated with uniform sampling in $[10^{-5}; 10^{-1}]$, and that the o objects were also uniformly pseudo-randomly assigned to a subset of p processors.

Because the time-to-solutions were approximately equal ($\approx$ 52s) as a result of $i \times k$ being equal to 20 in both cases, the x-axis of Figure 2.3 was re-normalized with respect to a common actual runtime of 52s, and the state at the end of each balancing iteration is shown. We note that in reality the time-to-solution in the $i = 10$ case was slightly higher than with $i = 2$, as a result of the latter requiring 11 I/O calls, in contrast with the 3 calls required by the former. We observe that not only time-to-solutions are identical, but that allowing for more gossiping rounds.

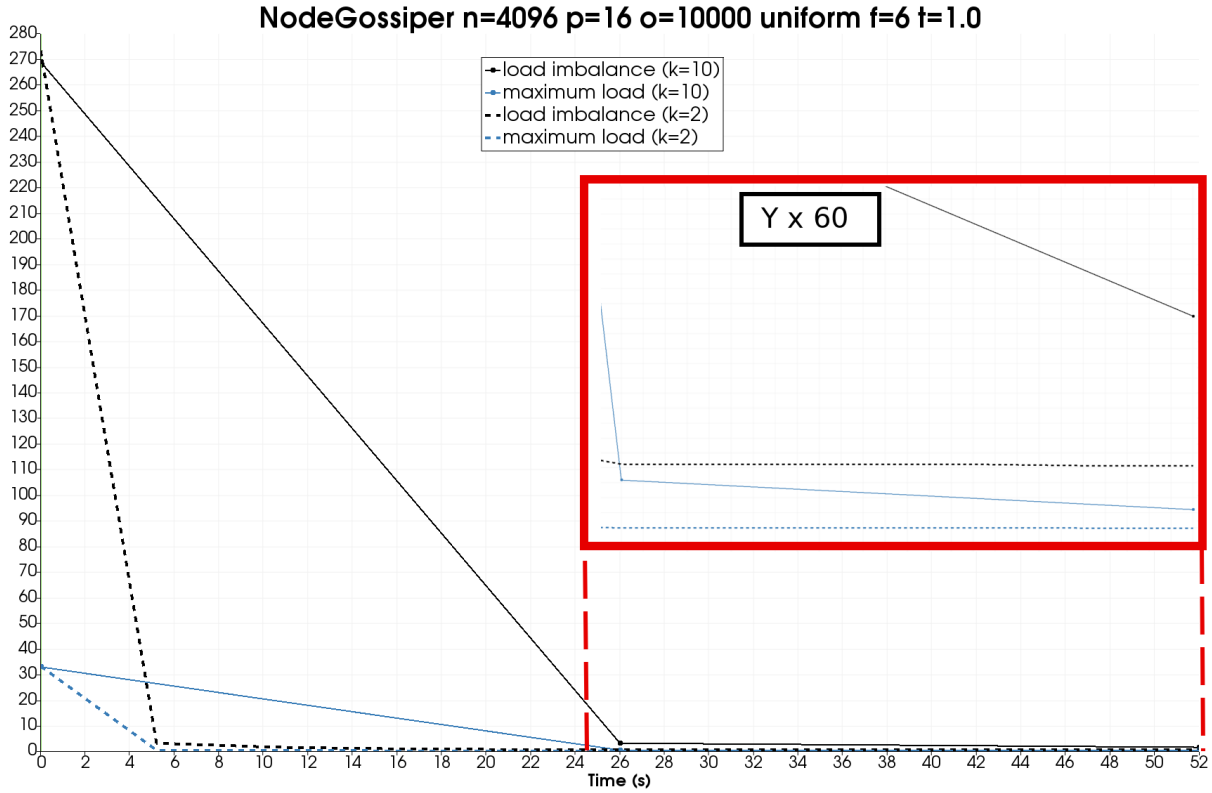

Figure 2.3: Plot of global imbalance ($\mathcal{I}$) and maximum load ($L_{\max}$) versus time normalized to 52s (average time to solution); plot ordinates are rescaled by a factor of 60 in the red inset for the second half of the time interval to emphasize convergence.

Chapter 3

Statistical Properties of Some Solvable Cases

The goal of this chapter is to examine the statistical properties of some cases for which they can be explicitly calculated, hereby providing the basis for reference non-regression testing as well as performance comparisons.

In particular, all observed load/processor distributions will be compared to an ideal *equi-distribution*, in which all objects are assigned in such a way that all processors in the collection $\mathbf{P}$ have identical load. We note that this does not require that all objects have the same load.

Respectively denoting $|L|$ and δ_i the total load across $\mathbf{P}$, and the Dirac delta function on processor $p_i \in \mathbf{P}$, we can denote $\mathcal{L} = \frac{|L|}{|\mathbf{P}|} \sum_{i=1}^{|\mathbf{P}|} \delta_i$ the equi-distribution of loads (ignoring the object dispatch details) on $\mathbf{P}$, for which we have the following statistics: $\min \mathcal{L} = \max \mathcal{L} = \overline{\mathcal{L}} = \frac{|L|}{|\mathbf{P}|}$, $\sigma^2_{\mathcal{L}} = \mu_{3,\mathcal{L}} = \mu_{4,\mathcal{L}} = 0$, and $\mathcal{I}_{\mathcal{L}} = 0$ for obvious reasons.

All observed load/processor distributions will have to be compared against this ideal baseline.

3.1 Dirac Distributions

By *Dirac distribution*, we mean here any case where all objects are assigned to a single processor in the non-empty collection $\mathbf{P}$. Without any loss of generality for the sake of statistical properties, we can assume that this processor is p_1, and we therefore consider the load distribution $\mathcal{L} = |L|\delta_1$. Also denoting $|\mathbf{P}|$ the cardinality of $\mathbf{P}$, we obviously have the following statistics for $\mathcal{L}$:

$$\begin{aligned} \min \mathcal{L} &= 0, \\ \max \mathcal{L} &= |L|, \\ \overline{\mathcal{L}} &= \frac{|L|}{|\mathbf{P}|}. \end{aligned}$$

Therefore, using the latter equation, we obtain the variance of $\mathcal{L}$ as:

$$\begin{aligned}\sigma^2_{\mathcal{L}} &= \frac{1}{|\mathbf{P}|}\sum_{i=1}^{|\mathbf{P}|}\left(|L|\delta_1(i) - \frac{|L|}{|\mathbf{P}|}\right)^2 \\ &= \frac{|L|^2}{|\mathbf{P}|}\left[\left(1-\frac{1}{|\mathbf{P}|}\right)^2 + \frac{|\mathbf{P}|-1}{|\mathbf{P}|^2}\right] \\ &= \frac{|L|^2(|\mathbf{P}|-1)}{|\mathbf{P}|^2} \underset{|\mathbf{P}|\to\infty}{\sim} \frac{|L|^2}{|\mathbf{P}|}.\end{aligned}$$

Concerning the third and fourth central moments of $\mathcal{L}$, we have on one hand:

$$\begin{aligned}\mu_{3,\mathcal{L}} &= \frac{1}{|\mathbf{P}|}\sum_{i=1}^{|\mathbf{P}|}\left(|L|\delta_1(i) - \frac{|L|}{|\mathbf{P}|}\right)^3 \\ &= \frac{|L|^3}{|\mathbf{P}|}\left[\left(1-\frac{1}{|\mathbf{P}|}\right)^3 - \frac{|\mathbf{P}|-1}{|\mathbf{P}|^3}\right] \\ &= \frac{|L|^3}{|\mathbf{P}|} \times \frac{(|\mathbf{P}|-1)^3 - |\mathbf{P}| + 1}{|\mathbf{P}|^3} \\ &= \frac{|L|^3\left(|\mathbf{P}|^2 - 3|\mathbf{P}| + 2\right)}{|\mathbf{P}|^3},\end{aligned}$$

and on the other hand:

$$\begin{aligned}\mu_{4,\mathcal{L}} &= \frac{1}{|\mathbf{P}|}\sum_{i=1}^{|\mathbf{P}|}\left(|L|\delta_1(i) - \frac{|L|}{|\mathbf{P}|}\right)^4 \\ &= \frac{|L|^4}{|\mathbf{P}|}\left[\left(1-\frac{1}{|\mathbf{P}|}\right)^4 + \frac{|\mathbf{P}|-1}{|\mathbf{P}|^4}\right] \\ &= \frac{|L|^4}{|\mathbf{P}|} \times \frac{(|\mathbf{P}|-1)^4 + |\mathbf{P}| - 1}{|\mathbf{P}|^4} \\ &= \frac{|L|^4\left(|\mathbf{P}|^3 - 4|\mathbf{P}|^2 + 6|\mathbf{P}| - 3\right)}{|\mathbf{P}|^4}.\end{aligned}$$

The above results allow us to compute the skewness and kurtosis of the distribution, when $|\mathbf{P}| > 1$, respectively as follows:

$$\gamma_{1,\mathcal{L}} = \frac{\mu_{3,\mathcal{L}}}{\sigma^3_{\mathcal{L}}} = \frac{|\mathbf{P}|^2 - 3|\mathbf{P}| + 2}{(|\mathbf{P}|-1)^{3/2}} \underset{|\mathbf{P}|\to\infty}{\sim} \sqrt{|\mathbf{P}|},$$

and

$$\gamma_{2,\mathcal{L}} = \frac{\mu_{4,\mathcal{L}}}{\sigma^4_{\mathcal{L}}} = \frac{|\mathbf{P}|^3 - 4|\mathbf{P}|^2 + 6|\mathbf{P}| - 3}{(|\mathbf{P}|-1)^2} \underset{|\mathbf{P}|\to\infty}{\sim} |\mathbf{P}|.$$

Finally, we observe that $\mathcal{I}_{\mathcal{L}} = |\mathbf{P}| - 1 \underset{|\mathbf{P}|\to\infty}{\sim} |\mathbf{P}|$.

The results above shall be used for statistical verification of idealized cases, as upper bounds on worst-case scenarii (paying attention to the fact that most statistical packages report *kurtosis excess*, i.e., $\gamma_2 - 3$).

3.2 Optimal Distributions

By *optimal distribution*, what is meant in what follows is the best possible load/processor distribution that may be achieved given the sets of processors $\mathbf{P}$ and of objects $\mathbf{O}$. It is not necessarily unique; in fact, except in the uni-processor case, it is not, because if one load distribution across $|\mathbf{P}| > 1$ processors is optimal, then any distribution obtained by exchanging all objects between two processors is also optimal.

3.2.1 Object Iso-Load Case

To start, we place ourselves in the idealize case where all objects $O_i \in \mathbf{P}$ have the same load $load(O_i) = \ell$. Although it is safe to assume that this ideal case will very seldom, if ever, be encountered in any real case, it is important that any proposed load-balancing algorithm be able to perform optimally in this case.

statistic	**optimal**	iteration	criterion `6`	criterion `6'`
$\min \mathcal{L}$	**100**	0	80	
		4	99	**100**
$\max \mathcal{L}$	**100**	0	123	
		4	116	**100**
$\sigma_{\mathcal{L}}$	**0.00**	0	9.18	
		4	2.34	**0.00**
$\mathcal{I}_{\mathcal{L}}$	**0.00**	0	0.23	
		4	0.16	**0.00**

Table 3.1: Load/processor statistics before and after 4 iterations of the original and modified algorithms, with 4 gossiping rounds, in an iso-load case where $|\mathbf{P}|$ divides $|\mathbf{O}|$.

Evidently, if $|\mathbf{P}|$ divides $|\mathbf{O}|$, then any equi-distribution is optimal and is obtained by assigning exactly $\frac{|\mathbf{O}|}{|\mathbf{P}|}$ object(s) to each processor in $\mathbf{P}$. We therefore present in Table 3.1 the respective results of the Grapevine algorithm with both original criterion `6` (left) and our modified criterion `6'`, for a case where $\mathbf{P} = 10^2$, $\mathbf{O} = 10^4$, $i = k = f = 4$, $t = 1.0$, and where the unit load is chosen to be $\ell = 1$. These results demonstrate that even with small numbers of fanout, gossiping round, and iterations, our alternate approach rapidly converges to the optimal solution whereas the original one does not, even with this simple academic case. This first baseline case is thus discriminating enough to be retained.

In order to extend the set of baseline cases, we now consider a slightly broader class of distributions, where all objects still have equal load ℓ, but where it is no longer required that $|\mathbf{P}|$ divide $|\mathbf{O}|$. In this context, denoting $q = \lfloor {}^{|\mathbf{O}|}/_{|\mathbf{P}|} \rfloor$ and $r = |\mathbf{O}| \bmod |\mathbf{P}|$, consider a distribution of q objects on $|\mathbf{P}| - r$ processors ($0 \leq r < |\mathbf{P}|$ by definition of the Euclidean division) and $q+1$ objects on the remaining r processors. All objects are indeed assigned because $r(q+1)+(|\mathbf{P}|-r)q = r+q|\mathbf{P}| = |\mathbf{O}|$. We thus have, if $r \neq 0$ (i.e., in the non-divisible case):

$$\begin{aligned} \min \mathcal{L} &= q\ell, \\ \max \mathcal{L} &= (q+1)\ell, \\ \overline{\mathcal{L}} &= \frac{|\mathbf{O}|}{|\mathbf{P}|}\ell, \end{aligned}$$

wherefrom we find that

$$\mathcal{I}_\mathcal{L} = \frac{q+1}{{}^{|\mathbf{O}|}/_{|\mathbf{P}|}} - 1 = \frac{q|\mathbf{P}| + |\mathbf{P}| - |\mathbf{O}|}{|\mathbf{O}|} = \frac{|\mathbf{P}| - r}{|\mathbf{O}|} > 0.$$

We note that the case where $r = 0$, we have instead

$$\mathcal{I}_\mathcal{L} = \frac{q}{{}^{|\mathbf{O}|}/_{|\mathbf{P}|}} - 1 = \frac{q|\mathbf{P}| - |\mathbf{O}|}{|\mathbf{O}|} = 0,$$

which amounts to the previously discussed distribution where $|\mathbf{P}|$ divides $|\mathbf{O}|$, that we already proved to be optimal.

Furthermore, as soon as $r \neq 0$, any other distribution assigning more than $q+1$ objects to at least one processor has a larger imbalance, because the denominator $\overline{\mathcal{L}}$ in $\mathcal{I}_\mathcal{L}$ is fixed. Therefore, a distribution $\mathcal{L}$ of $|\mathbf{O}|$ objects with identical load ℓ across $|\mathbf{P}|$ processors is optimal if and only if it can be written (with no loss of generality, up to a re-indexing of processors) as follows:

$$\mathcal{L} = (q+1)\ell \sum_{i=1}^{r} \delta_i + q\ell \sum_{i=r+1}^{|\mathbf{P}|} \delta_i.$$

Because $\delta_i \delta_j = \delta_i$ if $i = j$, and 0 otherwise, we have

$$\mathcal{L}^2 = (q+1)^2\ell^2 \sum_{i=1}^{r} \delta_i + q^2\ell^2 \sum_{i=r+1}^{|\mathbf{P}|} \delta_i$$

as all cross-products vanish, and thus

$$\overline{\mathcal{L}^2} = (q+1)^2\ell^2 \sum_{i=1}^{r} \overline{\delta_i} + q^2\ell^2 \sum_{i=r+1}^{|\mathbf{P}|} \overline{\delta_i} = (q+1)^2\ell^2 \frac{r}{|\mathbf{P}|} + q^2\ell^2 \frac{|\mathbf{P}| - r}{|\mathbf{P}|} = \frac{\ell^2}{|\mathbf{P}|}(r + 2qr + q^2|\mathbf{P}|).$$

This allows us to compute the variance of $\mathcal{L}$, as follows:

$$\begin{aligned}\sigma_{\mathcal{L}}^2 &= \overline{\mathcal{L}^2} - (\overline{\mathcal{L}})^2 \\ &= \frac{\ell^2}{|\mathbf{P}|}\Big(r + 2qr + q^2|\mathbf{P}|\Big) - \frac{|\mathbf{O}|^2}{|\mathbf{P}|^2}\ell^2 \\ &= \frac{\ell^2}{|\mathbf{P}|^2}\Big[r|\mathbf{P}| + 2qr|\mathbf{P}| + q^2|\mathbf{P}|^2 - (q|\mathbf{P}| + r)^2\Big] \\ &= \frac{\ell^2 r(|\mathbf{P}| - r)}{|\mathbf{P}|^2} \leq \frac{\ell^2}{4},\end{aligned}$$

with equality if and only if $\mathbf{P} = 2r$. We note that in the particular case where $r = 0$, we indeed retrieve the null variance result.

We thus tested the respective results of the Grapevine algorithm with both original criterion `6` and our modified criterion `6'` when $r \neq 0$, to assess how those respectively fared as compared to optimal distributions. Because the algorithm endowed with the original criterion has already not been able to obtain an optimal distribution in the easier $r = 0$ case, it is safe to expect that it will also not attain an optimal one in this more complicated case. But it is interesting to test whether the algorithm equipped with alternate criterion instead still performs optimally in the case.

statistic	**optimal**	iteration	criterion `6`	criterion `6'`
$\min \mathcal{L}$	**39**	0	23	
		4	**39**	**39**
$\max \mathcal{L}$	**40**	0	61	
		4	45	**40**
$\sigma_{\mathcal{L}}$	**0.2421**	0	6.182	
		4	0.5192	**0.2421**
$\mathcal{I}_{\mathcal{L}}$	**0.024**	0	0.5616	
		4	0.152	**0.024**

Table 3.2: Load/processor statistics before and after 4 iterations of the original and modified algorithms, with 4 gossiping rounds, in an iso-load case where $|\mathbf{P}|$ does not divide $|\mathbf{O}|$.

For instance, keeping the same example as above, except that now $\mathbf{P} = 16^2$, which results in $r = 16 \neq 0$ (and $q = 39$), and the same initial seeding of the sampler, we obtained the results presented in Table 3.2, compared against the optimal statistics computed from the preceding theoretical results. We thus observe that once again, while (as expected) the original algorithm fails to discover an optimal distribution, our version incorporating modified criterion `6'` succeeds at doing so.

We therefore propose that the two object iso-load cases (with $r = 0$ and with $r \neq 0$ be systematically used as baseline test cases for all distributed load balancers.

3.2.2 Iso-Load Approximations

It is a natural question to wonder whether the optimality results obtained in the idealized iso-load case may also be used as approximations of the expected load-balancing results when the objects are of relatively homogenous sizes.

statistic	iteration	$\varepsilon = 0$	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-2}$	$\varepsilon = 10^{-1}$	$\varepsilon = 1$
$\min \mathcal{L}$	0	23	23.00	22.99	23.20	23.45
	4	**39**	38.99	38.07	38.11	37.49
$\max \mathcal{L}$	0	61	61.00	61.03	60.89	66.37
	4	**40**	40.00	40.00	40.05	40.28
$R_{\mathcal{L}}$	0	38	38.00	38.04	37.69	42.92
	4	**1**	1.01	1.93	1.94	2.79
$\sigma_{\mathcal{L}}$	0	6.182	6.182	6.179	6.217	7.516
	4	**0.2421**	0.2405	0.2556	0.3748	0.3804
$\mathcal{I}_{\mathcal{L}}$	0	0.5616	0.5616	0.5624	0.5586	0.6974
	4	**0.024**	0.02399	0.02393	0.02525	0.03020

Table 3.3: Load/processor statistics before and after 4 iterations of the original and modified algorithms, with 4 gossiping rounds, when $|\mathbf{P}|$ does not divide $|\mathbf{O}|$, when uniform random noise of increasing magnitude is added to the **iso-load case**.

For instance, *ceteris paribus* as compared to the case of Table 3.2, but now with $load(O_i) \sim \mathcal{U}(1-\varepsilon; 1+\varepsilon)$, we obtain the results presented in Table 3.3, with one experiment for each of the reported values of ε (now only using alternate criterion `6'`). At first glance, we may be tempted to posit that the iso-load approximations provide good approximations unless the object loads are allowed to fluctuate within a few percentage points about the mean ℓ, with a progressively worsening spread between processor loads, as indicated by the range statistic $R_{\mathcal{L}}$.

However, this is a false impression because the results in Table 3.3 were obtained with $i = 4$ iterations of the algorithm: this number, albeit sufficient to reach an optimal distribution in the iso-load case, results in stopping the convergence to a better-balanced solution in the other cases, which translates into rejection rates that have not converged to 0. This observation is hinting at the fact that, when $\varepsilon \neq 0$, the algorithm at its fourth iteration continues to improve the results in the $\|\cdot\|_\infty$ sense, and more markedly as ε increases. For instance, with $\varepsilon = 1$ we still observe transfer acceptance rates higher than 10% at iteration number 4.

And indeed, as illustrated in Table 3.4, when we let the algorithm run through 100 iterations, we observe wholly different results: at that point all cases have converged, with rejection rates of 0% since many iterations: specifically, we obtain *better* load-balancing results, as the noise added to the atomic load ℓ provides flexibility in how the algorithm may compensate

statistic	iteration	$\varepsilon = 0$	$\varepsilon = 10^{-3}$	$\varepsilon = 10^{-2}$	$\varepsilon = 10^{-1}$	$\varepsilon = 1$
$\min \mathcal{L}$	100	**39**	39.00	38.95	39.07	39.07
$\max \mathcal{L}$	100	**40**	40.00	39.94	39.54	39.37
$R_{\mathcal{L}}$	100	**1**	1.00	0.99	0.47	0.30
$\sigma_{\mathcal{L}}$	100	**0.2421**	0.2403	0.2257	0.2581	0.03111
$\mathcal{I}_{\mathcal{L}}$	100	**0.024**	0.02384	0.02243	0.01205	0.006778

Table 3.4: Load/processor statistics after 100 iterations of the original and modified algorithms, with 4 gossiping rounds, when $|\mathbf{P}|$ does not divide $|\mathbf{O}|$, when uniform random noise of increasing magnitude is added to the **iso-load case**.

for the fact that $|\mathbf{P}|$ does not divide $|\mathbf{O}|$. In the strict iso-load case, there is just no way to reduce the load imbalance between the processors that contain q objects, and those that contain $q + 1$; in contrast, as ε increases, the runtime finds more opportunities to displace objects to better fill that gap. This is confirmed as the spread between the most underloaded and overloaded processors decreases as well.

From these observations we can can conclude that the iso-load formulas of §3.2.1 are *not* good predictors of the final outcome of the algorithm when it is allowed to run to convergence in non-iso-load cases; however, they remain decent predictors of what is to be expected in a non-iso-load case when the number of iterations is clamped to that which is sufficient for convergence in the iso-load case. This will also therefore provide a good baseline testing case.

This page intentionally left blank.

Chapter 4

Application to a Small Yet Real Case

In this chapter, we compare the results of the original Grapevine load-balancing algorithm, with those obtained with our modified version therefore, when applied to a real case whose input data, rather than being pseudo-randomly generated as in the previously discussed cases, was provided by an object mapping output by a VT-based simulation.

We also illustrate the somewhat non-intuitive object transfers that were performed by our algorithm in order to attain a near-optimal solution.

4.1 Experimental Setting

The case used here is that of a run performed on 8 processors using the Virtual Transport (VT) asynchronous many mapping was output every several time steps, using a data exchange format (`.vom`) that we created for that purpose. The NodeGossiper is endowed with both a reader and writer for this format, allowing it to compute a hopefully better-balanced object mapping than the one it was handed by the simulation code.

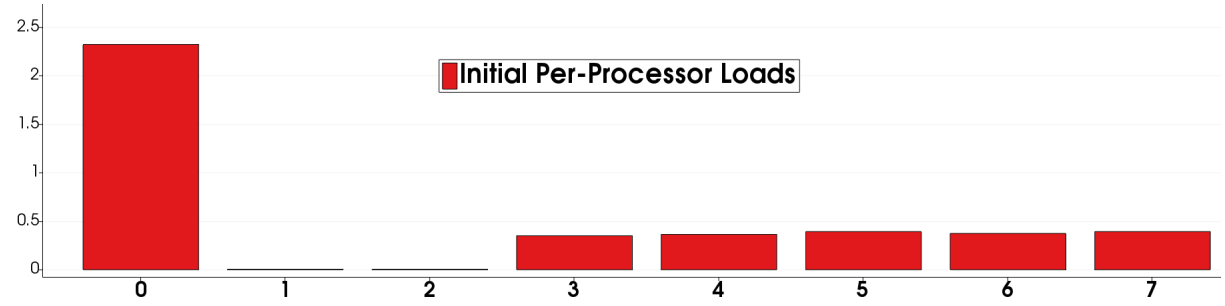

Figure 4.1: Histogram of the initial per-processor loads.

The experiment we are presenting used an object mapping read from the `.vom` files, whose aggregrated per-processor times are illustrated in Figure 4.1: a cursory visual inspection of this histogram readily indicates a poorly-balanced initial distribution.

In order to provide a quantitative assessment of the distributions obtained after having applied the NodeGossiper (with either of its two of variants), with respect to the initial situation, some key statistical properties are shown in Table 4.1: the first row provides the

$\mathcal{D}$	$\lvert\mathcal{D}\rvert$	$\min \mathcal{D}$	$\overline{\mathcal{D}}$	$\max \mathcal{D}$	$R_{\mathcal{D}}$	$\sigma_{\mathcal{D}}$	$\gamma_{1,\mathcal{D}}$	$\gamma_{2,\mathcal{D}}$	$\mathcal{I}_{\mathcal{D}}$
O	100	0.00026703	0.042076	0.30404	0.30377	0.074587	2.8855	9.8574	N/A
$\mathcal{L}$	8	0.004673	0.52595	2.3209	2.3162	0.69632	2.030	5.5987	3.4128

Table 4.1: Object mapping statistics of the initial object/processor load distribution of a VT-based simulation on 8 ranks.

values for the object times themselves, while the second line presents those for our variable of interest, the per-processor load $\mathcal{L}$. These statistics indicate in particular overall load imbalance that is almost half the size of the rank set, with the most heavily loaded rank being assigned 500 times more work that the least loaded one. We shall now examine how both versions of the algorithm perform in this case.

4.2 Comparative Results

We now compare the results obtained with the two different implementations (with original criterion and `6` vs. with modified criterion `6'`) by using in both cases the following input parameters: $i = 5$, $k = f = 4$, and an overload threshold $T = 1_0$, and identical seedings of the pseudo-random number generator used by the samplers in the two phases of the algorithm.

criterion	$\min \mathcal{L}$	$\max \mathcal{L}$	$R_{\mathcal{L}}$	$\sigma_{\mathcal{L}}$	$\gamma_{1,\mathcal{L}}$	$\gamma_{2,\mathcal{L}}$	$\mathcal{I}_{\mathcal{L}}$
`6`	0.28542	1.7508	1.4654	0.46454	2.2364	6.0655	2.3288
`6'`	0.49539	0.57315	0.077759	0.024726	0.77394	2.3940	0.089739

Table 4.2: Object mapping statistics obtained after 5 iterations of the load-balancing algorithm with both version of the decision criteria (`6` vs. `6'`) when $k = f = 4$.

The results obtained for each experiment presented in Table 4.2. Cardinalities and averages are omitted as they are identical to those indicated in the last row of Table 4.1, as both are invariant because we do not allow the number of ranks (nor of course the total load) to change during the load-balancing process.

In particular, we observe that the original version of the algorithm only marginally improved the results, with a global imbalance $\mathcal{I}$ keeping the same order of magnitude pre- and post-load-balancing; in contrast, when using our variant `6'`, the same was improved more than thirty-fold. Furthermore, we can see that the range R in the latter case has a value (0.077759) that is almost equal to that of the standard deviation of object times (0.074587): in other words, our modified criterion allows the load-balancing to reach near-optimality in this case as well, with modest values of i, k, and f. This quantitative comparison confirms what we had demonstrated before, from first principles and analytical cases.

4.3 Prescribed Object Moves

In this section, we use the VOM viewer that we created to generate to create graphs representing the object move counts prescribed by the load-balancer. In these graphs, vertices represent the processors, while edges represent object moves, and are endowed with two attributes: directed and undirected moves, i.e., respectively, the number of object transfers in each direction, and the total count of movements in both directions. While the edge coloring scheme indicates the latter, overlaid arrows using the same color scale indicate the prescribed traffic in each direction. In particular, two-way traffic is indicated by pairs of opposing arrows, whereas in the case where object moves are prescribed in a single direction between two processors, a single overlaid arrow will indicate that – it will also, as a result, have the same color as the edge over which it is drawn.

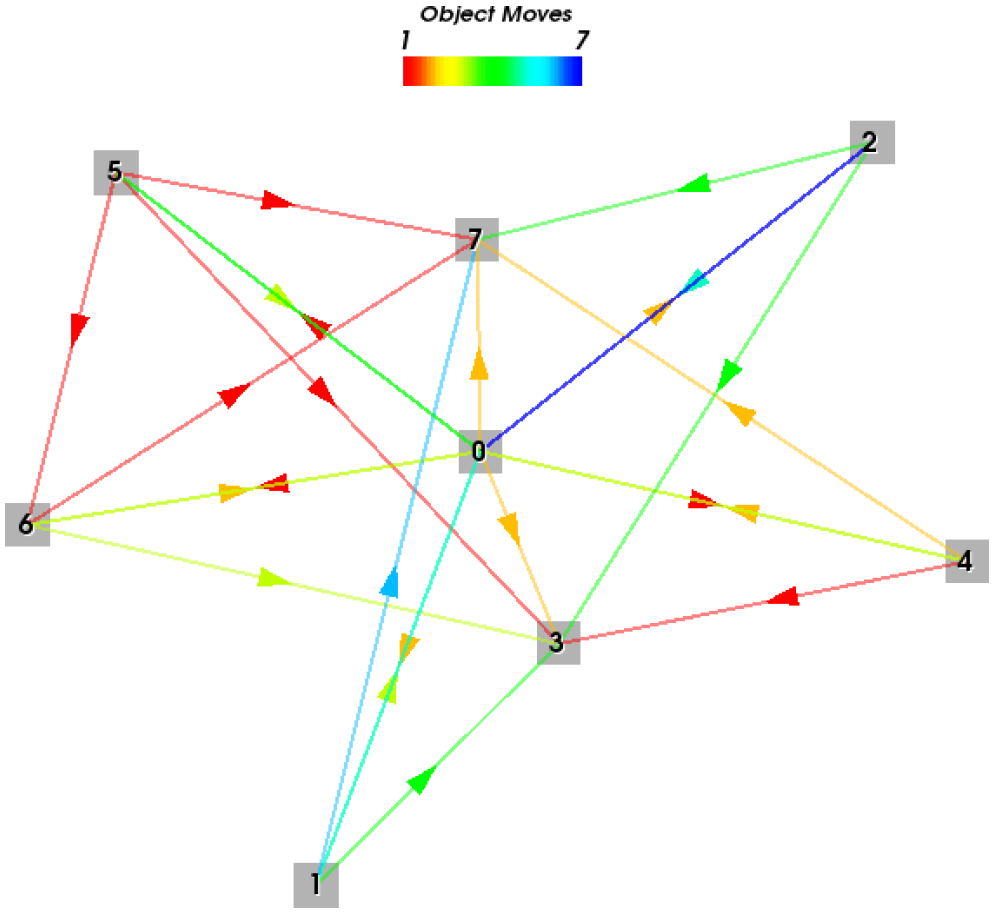

Figure 4.2: Graph representing the number of inter-rank object moves prescribed by the load-balancer after 5 iterations of our improved version (with criterion `6'`) when $k = f = 4$.

Figure 4.2 supports the intuition in that we observe that, as one would expect by looking at the original object/processor distribution in Figure 4.1, the most heavily loaded processor (0) is the one that offloads the most objects to other ranks; in fact, every other processor

shall receive at least one object from processor 0, with processor 2 (which was minimally loaded to begin with) will receive the most. On the other hand, we observe that marginal load adjustments occur between the 5 last ranks (3–7) which in Figure 4.1 form a cluster of similarly-loaded processors.

What is less intuitive is that the two processors which were essentially devoid of significant load initially (1 and 2) are nonetheless prescribed to ship some of their respective objects to other ranks; the most surprising being maybe that both processors are even asked offload several of their respective objects onto the most heavily loaded of their peers (0). Upon closer inspection, we see that this apparent paradox is easily explained by the positive skew in the object times distribution (cf. Table 4.1) which indicates that there are relatively more object with small than with large times, with a few outliers in the latter category pulling the mean to the right. In other words, our variant of the algorithm not only performs gross load adjustments, but also keeps moving small pieces of work until the distribution is near-optimally distributed.

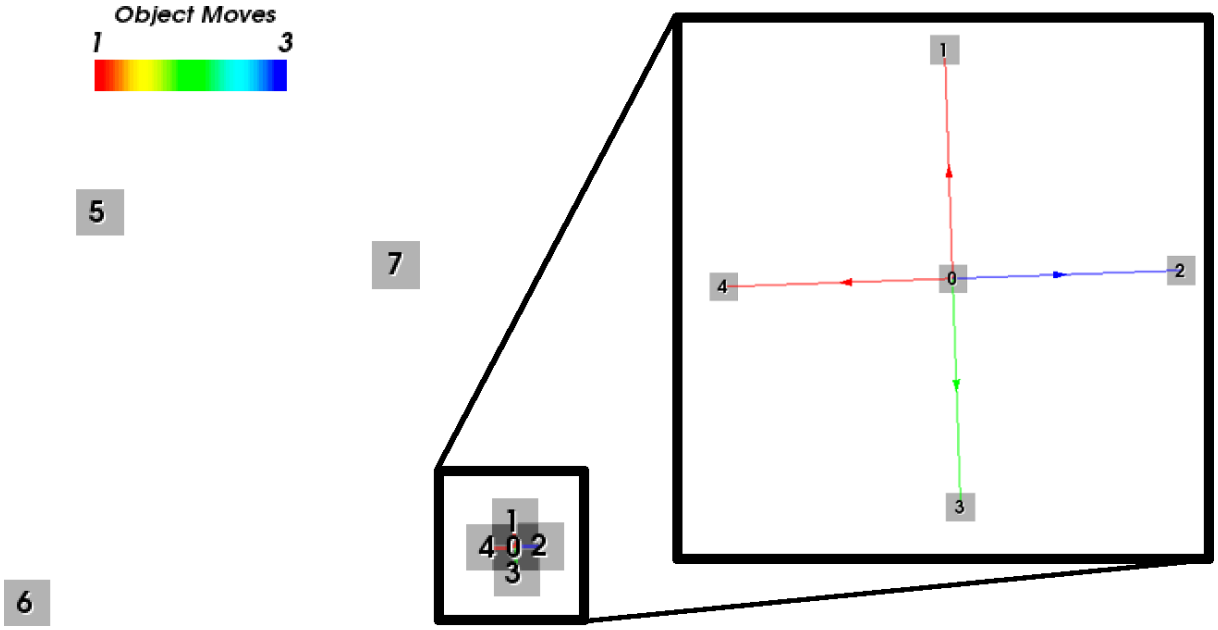

Figure 4.3: Graph representing the number of inter-rank object moves prescribed by the load-balancer after 5 iterations of the original algorithm (with criterion 6) when $k = f = 4$. Note that the cluster of ranks $\{0; 1; 2; 3; 4\}$ is enlarged in the upper right corner for legibility.

In contrast, as illustrated in Figure 4.3, the original algorithm becomes quickly trapped in a local, very sub-optimal minimum. Indeed, once the most heavily loaded processor (0) has shipped all possible objects to the least loaded once without making them overloaded, the 3 processors whose initial load is closest to the mean $\overline{\mathcal{L}} \approx 0.52595$, albeit slightly underloaded, are forced by the same criterion 6 to rejected all proposed object transfers that can make them overloaded, even slightly. This results in a locked configuration where no global improvement can further occur, although the global imbalance $\mathcal{I}$ remains high.

Chapter 5

Conclusion

Because of the results presented in this report, both theoretical and experimental, we propose to use exclusively our variant of the algorithm in the future VT load-balancer implementation.

Consider the case where $\mathbf{P} = \{p_1; p_2\}$, $\mathbf{O} = \{O_1; O_2\}$, and $load(O_1) = load(O_2) = \ell$ (in the sense of time needed to perform these tasks), and denote as follows the 4 possible object/processor mappings: $m_{ij} = \{O_1 \rightarrow p_i; O_2 \rightarrow p_j\}$. Both m_{12} and m_{21} have null imbalance, whereas for both m_{11} and m_{22} $\mathcal{I} = 1$. In other words, in terms of object/processor balance, as has been the scope of this entire report, of the 4 possible distributions, exactly 2 are optimal and occur when exactly one object in $\mathbf{O}$ is assigned to each processor in $\mathbf{P}$.

But now, consider the additional piece of information whereby an some data must be communicated between O_1 and O_2 before the work (with time ℓ) can be completed by each. For the sake of simplicity, assume and that it takes no time to perform this data exchange when both objects reside on the same rank, whereas it takes a time $\tau > 0$ to do it when they do not. When factoring in this new parameter, we see that the total time T that is required to perform the entire work (i.e., executing all objects to completion) takes on the following values:

$$T(m_{11}) = T(m_{22}) = 2\ell \quad ; \quad T(m_{12}) = T(m_{21}) = \ell + \tau.$$

Evidently, depending on the relative values of ℓ and τ, the best use of the available resources is not necessarily and optimal object/processor distribution. Specifically, these two concepts coincide in our example if, and only if, $\tau < \ell$.

How the taking into account of the inter-object communication costs will affect the overal compute time, and modify what is considered an optimal distribution, will be the focus of our future work.

This page intentionally left blank.

References

[1] K Mani Chandy and Jayadev Misra. How Processes Learn. *Distributed Computing*, 1(1):40–52, Springer, 1986.

[2] Edsger W Dijkstra and Carel S Scholten, Termination Detection for Diffusing Computations. *Information Processing Letters*, 11(1):1–4, Elsevier, 1980

DISTRIBUTION:

1	MS N/A	Philippe P. Pébaÿ, 08753
1	MS N/A	Jonathan Lifflander, 08753
1	MS 0899	Technical Library, 8944 (electronic copy)


Sandia National Laboratories