J Sched
DOI 10.1007/s10951-009-0138-4

Adaptive statistical scheduling of divisible workloads in heterogeneous systems

Horacio González-Vélez · Murray Cole

Abstract This article presents a statistical approach to the scheduling of divisible workloads. Structured as a task farm with different scheduling modes including adaptive single and multi-round scheduling, this novel divisible load theory approach comprises two phases, calibration and execution, which dynamically adapt the installment size and number. It introduces the concept of a generic installment factor based on the statistical dispersion of the calibration times of the participating nodes, which allows automatic determination of the number and size of the workload installments. Initially, the calibration ranks processors according to their fitness and determines an installment factor based on how different their execution times are. Subsequently, the execution iteratively distributes the workload according to the processor fitness, which is continuously re-assessed throughout the program execution. Programmed as an adaptive algorithmic skeleton, our task farm has been successfully evaluated for single-round scheduling and generic multi-round scheduling using a computational biology parameter-sweep in a non-dedicated multi-cluster system.

H. González-Vélez (✉)
Robert Gordon University, School of Computing,
Aberdeen AB25 1HG, UK
e-mail: h.gonzalez-velez@rgu.ac.uk

H. González-Vélez
Digital Technologies, IDEAS Research Institute, Aberdeen, UK
e-mail: horacio@acm.org

M. Cole
University of Edinburgh, School of Informatics,
Edinburgh EH8 9AB, UK
e-mail: mic@inf.ed.ac.uk

Keywords Divisible load theory · Divisible workloads · Scheduling · Task farm · Algorithmic skeletons · Structured parallelism · Parallel patterns · Parallel processing

1 Introduction

The generic process of mapping large groups of totally independent tasks with similar algorithmic complexity to distinct computational nodes is known as the scheduling of divisible workloads. Having been augmented with a schematic language and network element modelling, it has become an important area of study in computer science widely recognised as divisible load theory (DLT). Being particularly suitable for the solution of several representative computational problems, DLT has been the subject of monographs (Bharadwaj et al. 1996; Drozdowski 1997), article surveys (Bharadwaj et al. 2003; Blazewicz et al. 1999; Robertazzi 2003), and an annotated bibliographic repository (Robertazzi 2008).

The tasks in a divisible workload are independently distributed among all participating nodes in order to satisfy certain criteria, typically to minimise the makespan. This independence makes DLT particularly suitable for use in distributed systems, as the amount of work assigned to a certain node—i.e. the number of tasks—can be adjusted according to the node and interconnection characteristics. Nonetheless, DLT poses different challenges in terms of the characteristics of the computing nodes, the system workload, and the network topology.

Modern distributed systems—in the form of grids, clouds, or large clusters—are typically composed of a multiplicity of network links and computing nodes with dynamic latencies and computation capabilities. As a result, parallel programs are required to adapt to the intrinsic heterogeneity of

Published online: 16 October 2009

the platform, adjusting to the resource usage and availability at a given moment. Unfortunately, traditional system-wide scheduling strategies concentrate on user loads, ergo coarse-grain parallelism, disregarding the application characteristics.

We propose to address this scheduling problem using a task farm. A task farm (TF) consists of a farmer process which administers a set of independent worker processes to concurrently execute a large number of independent tasks, collectively comprising a *divisible workload*. In a traditional TF, each worker is allocated to a dedicated processor in a parallel machine, the computation of each element in the workload is independent and does not generate the same amount of work. The TF aims at fairly distributing the workload to avoid worker starvation and farmer node contention while minimising communication in order to produce the best load-balancing.

Different TF implementations assign distinct *task sizes*[1] to workers based on a certain *scheduling* method. Scheduling variants are typically classified by the number of *rounds* or *installments* in which the total workload is distributed.

Multi-round Scheduling. In its canonical form, the TF task-to-node mapping is based on a self-scheduled work queue (Hagerup 1997), where the farmer supplies one task to any available worker at a given time. After processing, a worker reports back to the farmer for the next unit of work or termination. For a given workload, each worker normally processes several tasks in multiple installments, constituting a *multi-round scheduling* schema. The work queue strategy provides an acceptable load balancing strategy for large workloads of undetermined size in dedicated systems with fixed network latency. The greedy nature of self-scheduling allows the assign-to-idle-node scheme to balance the system load over time. The generalisation of the work queue model allocates more than one task per round and takes into account variable network latency, effectively distributing small chunks of the workload in a greedy fashion.

Single-round Scheduling. In contrast to multi-round scheduling, this mode distributes the entire workload among the workers in one installment. Here, the task size is statically estimated at once to minimise idle time and ensure that minimal scheduling is required from the farmer's side. This is particularly relevant to fixed-size workloads in dedicated homogeneous systems.

In fact, as the scheduling deals out the workload, it ultimately determines the execution time on a per-node basis. The DLT optimality principle (Bharadwaj et al. 1996) states that, in order to minimise the makespan, all workers should stop their processing at the same time, otherwise there exists a better workload distribution. Nonetheless, the generic solution to the optimal scheduling of divisible workloads is proven to be NP-hard (Drozdowski and Lawenda 2008; Yang et al. 2007) and, therefore, remains an actively-studied, open-ended problem (Beaumont et al. 2005). In particular, non-dedicated heterogeneous systems pose an increased challenge (Beaumont et al. 2003), as the farmer is required to adapt the task size assigned to workers because:

– The underlying architecture can maintain multiple communication links between the farmer and workers with different bandwidths and latencies.
– The workers and the farmer can run on non-dedicated nodes with distinct background workloads in a distributed environment.

Hence, as opposed to off-line scheduling where the node resources and/or application characteristics are given in advance, it is arguable that an adaptive scheduling approach should not require previous knowledge of the task nature and the underlying infrastructure; or be constrained to a certain number of installments, nodes, or tasks.

Nevertheless, scant research has been devoted to the adaptive exploitation of the structure of a parallel application to improve the overall resource usage. Since the tasks in a divisible workload must be grouped in order to minimise communication costs in a distributed system, little attention is paid to partitioning using the application structure. We argue that the intrinsic coordination characteristics of an algorithmic skeleton place this paradigm in a preponderant position to explore workload scheduling. Based on the central premise of application adaptiveness to resource availability, we would like to research their actual correlation and provide an online scheduling methodology to enable a divisible workload to conform to the heterogeneity of a large distributed system.

1.1 Contribution and structure

This work significantly extends our initial findings on single-round scheduling (González-Vélez 2006) and parameterisable skeletal task farms (González-Vélez 2005), by providing a comprehensive statistical online framework to automatically schedule divisible workloads based on the dispersion of the participating nodes and size of the workload. Being application-agnostic and parameterisable, our approach addresses the multi-round scheduling case by defining an installment factor which dynamically quantifies the number of rounds using the number of tasks in the workload and the system circumstances. The single-round scheduling case is reduced to a special case for systems with low dispersion, where all participating nodes are equally able to process tasks, given their load conditions and processing capabilities.

The underlying assumptions are that the workload is embarrassingly parallel, i.e., all tasks are independent and each

[1]N.B. It is a common practice to refer to the number of tasks to be executed by a given worker/node as 'task size'.

task has similar computational complexity. The farmer and each worker process are presumed to be mapped to different nodes of a heterogeneous distributed system. Our approach is relevant to single-level tree architectures and, consequently, has been empirically evaluated on a multi-cluster with Ethernet interconnections.

This paper is structured as follows. Firstly, we examine other pertinent approaches to the scheduling of divisible workloads in heterogeneous systems and position our contribution accordingly. Secondly, we describe the relevant concepts of our methodology from a generalised multi-round approach, considering the single-round as a special case and introducing the concept of a generic installment factor based on the dispersion of the calibration times of the participating nodes. Thirdly, we present some experimental results using a parameter-sweep application for the estimation of calcium concentration, carried out on a non-dedicated heterogeneous multi-cluster system. Finally, we conclude with a discussion of the relevance of our work.

2 Related work

Historically preceding the advent of DLT, the distribution of independent atomic operations among processors has been analysed in the scheduling of parallel loops. Static strategies include isometric chunks where the overall set of operations, or a fixed subset, is equally divided among participating processors (Kruskal and Weiss 1985), and self-schedule workqueue where the distribution is unitary (Hagerup 1997). As a result of the evolving variation in the complexity of the operations and/or the load of the processors, dynamic loop scheduling strategies utilise resource awareness indicators to guide the number of operations assigned to a processor. Polychronopoulos and Kuck (1987) initially suggest the use of self-guided parallel loop scheduling, a methodical approach employing a decreasing number of operations per chunk based on the loop index in order to reduce the load imbalance. Safe self-scheduling augments such an approach with expected execution times, obtained through profiling or previous runs, to determine a variable number of operations (Liu et al. 1994). Adaptive factoring methods employ historical execution times of certain loop iterations, on a per processor basis, to adjust the number of operations delegated to a processor (Cariño and Banicescu 2008). These methods arguably supersede the traditional batch-oriented factoring, where a processing batch is determined through a fixed ratio of pending iterations and then divided equally among participating processors. Comparatively, our approach incorporates performance-based adjustment as in adaptive factoring and decreasing chunk size as in self-scheduling. Nevertheless, it also enhances such concepts by defining a dispersion-based installment factor, which determines the initial division of the workload, and a continual feedback process which maintains an acceptable task distribution among the nodes.

Moreover, it is widely acknowledged that one of the major challenges in large heterogeneous distributed systems is the prediction and improvement of performance. Such systems are characterised by the dynamic nature of their heterogeneous components, due to shifting patterns in background load which are not under the control of the individual application programmer. In principle, it is expected that efficient intra-application scheduling must be aware of the system conditions, and adapt their execution according to variations in the available computation and communication resources. The challenge is, therefore, to produce and support scheduling policies which can respond automatically to this variability.

From a more DLT-oriented perspective, abstract models for the scheduling of divisible workloads in heterogeneous systems have provided near-optimal theoretical solutions to particular cases. Banino et al. (2004) have developed a polynomial solution for the steady-state case, where all processing capabilities, applications requirements, and communication links are known in advance. The Uniform Multi-Round algorithm assumes that every node receives decreasing, fixed task sizes in every round and provides an approximation to the optimal number of rounds by minimising the application makespan in a simulated environment (Yang et al. 2005). Drozdowski and Lawenda (2007) tackle the problem as an optimisation of the application makespan but relax the assumption on fixed task sizes, approximating the solution via branch-and-bound and genetic algorithms on a simulated heterogeneous environment.

The online scheduling of divisible workloads therefore requires the dynamic generation of estimators to determine the task sizes and, ideally, the number of rounds. Ghose et al. (2005) propose the use of probing—the execution of a sample number of tasks on participating nodes to approximate the node processing and communication conditions—in order to distribute the tasks among the participating nodes accordingly. Although different research groups have included variations to probing in their works (Comino and Narasimhan 2002; Legrand et al. 2008; Li et al. 2005; van der Raadt et al. 2005; Viswanathan et al. 2007), effectively fostering resource-awareness in their online scheduling methods, our calibration approach is application-agnostic as it does not require any previous knowledge or performance figures for the actual application at hand.

Thus, the novelty of our work lies in

- the introduction of a dispersion-based installment factor to guide a decreasing chunk size;
- the use of application-agnostic calibration (probing) and execution routines to keep the initial performance assertions current; and

– its system infrastructure orientation as we do not rely on simulators, dedicated configurations, or performance estimators to model the general system, particularly to characterise the background load in terms of its job arrival rate.

3 Adaptive task farming

Adequate scheduling rests on the premise that the workload can be optimally distributed to the nodes with the most convenient resources for a given application, so it is crucial to be able to automatically enable an application to cope with resource variability. Our approach intends to optimise the application performance from a non-invasive systems infrastructure standpoint, using real resource measurements and application times.

At the core of our adaptive task farm are a calibration algorithm and an execution algorithm, which can be instantiated into different scheduling methods for assigning task sizes to different nodes according to their capacity, at once (single-round scheduling), or in several installments (multi-round scheduling).

Worker resources are quantified—at a given time on a certain system topology from an application-specific perspective—by means of a *fitness index* F. Defined during the calibration phase, F is to be used by the TF to determine the task size on a per node basis and, consequently, define the TF scheduling. Moreover, in the generic multi-round scheduling, its value is also adjusted during execution.

A task farm can be symbolically represented as $TF = \langle I, O, f \rangle$, where I is the input, O is the output, and f is the processing function. A worker executes a task by mapping f into a subset of I, computing a subset of O, and then reporting back to the farmer for the next unit of work or termination.

Let S denote the workload assigned to the farmer during the current round (thus for single round and the first round of multi-round schedules, $S = |I|$, the number of tasks in I), and N the number of participating workers (typically $N \ll S$). Thence, our objective is to calculate $\alpha_\imath$, the task size for each worker:

$$\alpha_\imath \ \forall \imath \in [1, N] \text{ subject to } \sum_{\imath=1}^{N} \alpha_\imath = S$$

We formally define F as the vector containing the relative fitness $F_\imath$ of each node:

$$\sum_{\imath=1}^{N} F_\imath = 1 \tag{1}$$

The actual values for $F_\imath$ are transient, as they periodically change according to the latest calibration of every node. Indeed, $F_\imath$ ought to be formally expressed as a function of time t, $F_\imath(t)$, where t is the time when the calibration snapshot is taken. However, since all decisions are local to each snapshot, we have simplified its notation by omitting t for readability purposes. This temporal behaviour of $F_\imath$ is further discussed in Sect. 3.2.2.

We can determine $\alpha_\imath$ as

$$\alpha_\imath = S \times F_\imath \quad \forall \imath \in [1, N] \tag{2}$$

Note that $\alpha_\imath$ is the total number of elements assigned to node $\imath$, and the key differentiator for the scheduling lies in how this amount is distributed. If distributed in one installment, then the scheduling will be considered single-round, otherwise it will be multi-round. Therefore, we can extend (2) to consider an *installment factor* k:

$$\alpha_\imath = \frac{S}{k} \times F_\imath \quad \forall \imath = 1, \dots, N \tag{3}$$

where $k, F_\imath \in \mathbb{R}$, $S \in \mathbb{N}$ and $0 < F_\imath \leq 1, k \geq 1$.
Since k and F are crucial to our approach, Sects. 3.1 and 3.2 discuss the calibration and execution phases respectively, with special emphasis on the determination of both parameters.

3.1 Calibration

During this phase, the N nodes are automatically calibrated with the execution of one element from the workload stored in I, the execution times are written to t, and the processed results are stored in O. Note that t is therefore the vector containing all $t_\imath$, the individual calibration nodes for each node. Then, F is computed using the inverse of the t, either direct or adjusted. These steps have been abstracted in Algorithm 1.

It is important to highlight that the calculation of F varies according to the calibration method, which can be:

- Times-only: The basic way to calculate F, times-only calibration defines F as a normalised decreasing function based on the inverse of $t_\imath$ for each $\imath$ node as shown in (4).

$$F_\imath = \frac{\frac{1}{t_\imath}}{\sum_{\jmath=1}^{N} \frac{1}{t_\jmath}} \tag{4}$$

- Statistical: F is determined by first employing a curve-fitting method for t, and then using the fitted t in (4).
  - Univariate Linear Regression: t is considered dependent on the processor availability.
  - Multivariate Regression: The processor availability and the network latency are considered independent and are employed to fit the t values.

While the overhead in the calibration is reduced, as this initial processing counts towards the overall processing, its complexity is still bound by the slowest node.

Algorithm 1: Calibration Algorithm for the Task Farm

```
Data: $I$: The input vector containing the
        divisible workload;
$O$: The output/result vector;
$f$: The worker function;
$N$: Number of workers;
$CM$: Calibration mode; /* Two options:
Times-only and Statistical                    */
Result: $\langle F_1, F_2, \ldots, F_N \rangle$; /* Each node
        individual fitness $F$                */

Calibrate $N$ nodes using $f$; /* Assumes $N \ll S$
*/
Store execution times in $t$ and results in $O$;
/* $t$ is a vector. Must wait for all nodes
   to complete before proceeding further
   (barrier)                                  */
if $CM = Statistical$ then
    /* Adjust $t$ via a curve-fitting method
       */
    Collect $a_\imath$, $\ell_\imath$, $bm_i$ $\forall \imath \in [1, N]$;
    Determine $a'_\imath$ $\forall \imath \in [1, N]$ ;
    if univariate regression then
        Calculate $t = f(a')$;
    else
        Calculate $t = f(a', \ell)$;
    end
end
forall $\imath \in [1, N]$ do
    /* Both statistical and times-only use
       this formula to calculate $F$          */
    Compute $F_\imath \leftarrow \frac{\frac{1}{t_\imath}}{\sum_{\jmath=1}^{N} \frac{1}{t_\jmath}}$ ;
end
Return $\langle F_1, F_2, \ldots, F_N \rangle$;
```

3.1.1 Statistical calibration

Statistical calibration has been widely used in the physical sciences to describe the use of measured physical variables in order to extrapolate a certain unknown via a series of mathematical transformations (Martens and Naes 1989).

In our case, the idea is to calculate the fitness of a certain node via the statistical extrapolation of its execution time, using the processor availability and the communication latency. This extrapolated fitness will ultimately determine the task size assigned to a node.

Given a certain node, its processor availability measures the processing fraction allocatable to a new process to be executed, while its communication latency is the time taken to receive a message from the farmer. Let $a_\imath$ and $\ell_\imath$ respectively be the processor availability and the communication latency for node $\imath$. Consequently, α, t, a, ℓ are vectors of size N which store the values for task size, execution time, availability, and latency. Supplied by a resource monitoring tool, the a and ℓ vectors contain measured physical values typically expressed as the CPU fraction allocatable to a new full-priority standard user process and the time in milliseconds to send a TCP message from the farmer to a certain node.

F is directly determined using t, and, transitively, so is α. As t is application-dependent, its value can be correlated with the resources available at a given time.

Such correlation can therefore be explored using:

– a only, uni-variate linear regression, or
– a and ℓ, multi-variate linear regression.

Univariate Linear Regression
Let us define $a'_\imath$, the scaled availability for worker $\imath$, as

$$a'_\imath = a_\imath \times rp'_\imath$$

$$rp'_\imath = \frac{bm_\imath}{\max_N(bm)}$$

where $rp_\imath$ is the relative performance of worker $\imath$, $bm_\imath$ is any known benchmark value for worker $\imath$, and $\max_N(bm)$ is the maximum $bm_\imath$ among N workers.
Using linear least-squares regression, we set a', the vector of $a'_\imath$ for the N workers as a predictor (independent variable) and allow t to be the dependent variable. Then, we attempt to fit a curve along the observed values in t using the regression function in (5).

$$t = c_0 + c_1 a' \tag{5}$$

Our objective is to assign fewer tasks to the workers which executed tasks more slowly and, in consequence, minimise the overall execution time. Hence, we calculate the F in (4), using the estimated (fitted) values t shown in expression (5).
Multivariate Linear Regression
Since processor availability is not necessarily the only determining factor, further exploration needs to take into account additional system parameters. In order to provide ground for discussion, Fig. 1 introduces the schematic representation of the relation between processor availability, communication latency, and execution times for the case study to be discussed in Sect. 5.1.
It is clear from Fig. 1 that the shortest execution times, represented by the darkest segments, tend to gravitate towards the right following the higher values of a, while the longest times are located in the upper left segment (lowest a). In this particular case, the strong implication of the trend is that the execution time on a given node is determined by

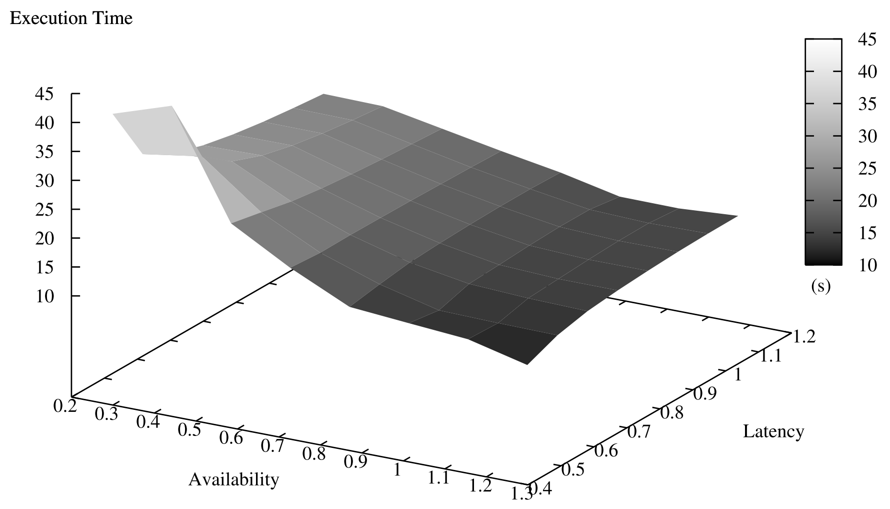


Fig. 1 The correlation between scaled availability (a'), network latency (ℓ), and execution times (t), where a' and ℓ are used as predictors and t as the dependent variable

the processor availability and is influenced, to a lesser extent, by the latency.

Using multi-variate linear least-squares regression, we set $a\prime$ and ℓ as the predictor vector within a matrix (X) and allow t to be the dependent vector. Then, similar to the univariate case, we use t as expressed in (6) to calculate F in (4).

$$t = c_0 + c_1\ell + c_2{a'}^2 \tag{6}$$

3.2 Execution

Single isometric installments are well suited to a dedicated homogeneous system, as its node processing capabilities are even. However, in a dynamic system with heterogeneous nodes, single installments should be determined using the node fitness and, in the case of multiple rounds, their actual number and size ought to be dynamically adjusted according to the system load and the prevalent fitness of the system nodes.

The execution phase for our TF can therefore be described as follows:

– if single-round scheduling, substitute $k = 1$ in (3) and, consequently, distribute the workload in one round, using F to calculate the task size for each node;
– otherwise, assume multi-round scheduling and calculate k based on the node dispersion. As the quotient $\frac{S}{k}$ in (3) implies multiple installments (if and only if $k > 1$), adapt the task size accordingly during the execution by refreshing F according to the most recent execution time for each node and the remaining amount of work to be completed.

3.2.1 Installment factor

One of the key issues when determining the task size is the initial number of tasks to be distributed. While single-round scheduling directly distributes the entire workload in the first round, generic multi-round scheduling is more complex. In the work queue case, it uses one task per node utilising as many tasks as nodes in the pool in the initial round, and continues in this fashion throughout the entire execution. Nonetheless, there is a potentially large number of possible combinations, which can use a larger number of tasks in each round and minimise the farmer–worker communication.

To this end, we have proposed to define a new concept: the *installment factor*. Denoted by k, this constant is intended to adaptively regulate the workload distribution in order to determine the installment size for a given worker in multi-round scheduling.

We determine k in terms of S, the workload, and the dispersion in the calibration times of the N nodes in the pool. This dispersion can be estimated by their coefficient of variation (CV), as represented in (7), and, as S can be easily conceived as a continually growing function for different problem instances, we can express k using (8).

$$\text{Given that } \bar{t} = \frac{1}{N}\sum_{\imath=1}^{N} t_i \quad \text{and} \quad \sigma = \sqrt{\frac{1}{N}\sum_{\imath=1}^{N}(t_i - \bar{t})^2},$$

$$CV = \frac{\sigma}{\bar{t}}, \tag{7}$$

$$k = \ln(S)^{CV} \tag{8}$$

Assuming that the differences in calibration times reflect not only the system heterogeneity but also its dynamism, a highly dynamic system will have a series of calibration times with a significantly large standard deviation, σ, and k will grow accordingly, while a steady system will have a negligible standard deviation and therefore k will approach 1, regardless of the input size. Nonetheless, for a given CV, the k will increase logarithmically on S. We have tacitly assumed that $S > e$, i.e., the divisible workload is composed

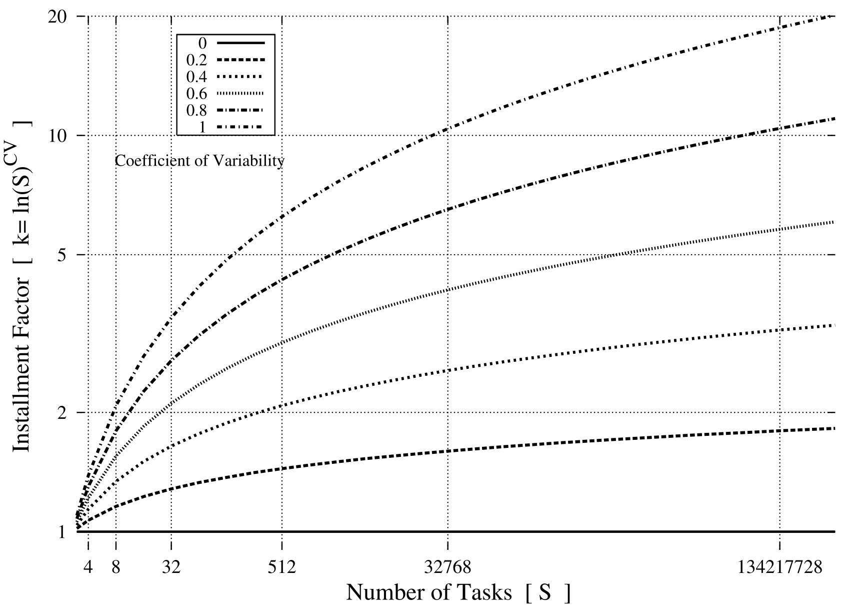


Fig. 2 The installment factor, k is a function of the number of tasks, S, and the coefficient of variability, CV, expressed as $k = \ln(S)^{CV}$. The six different lines of k are delineated by the variation of CV from 0 to 1 in intervals of size 0.2

of at least three tasks. The behaviour of k for different values of S and CV is plotted in Fig. 2.

We would like to emphasise that calculating k in this generic way relieves the programmer from statically defining the best scheduling, as the system automatically provides the most suitable number of rounds according to the dispersion in the system and the application at hand. Single-round scheduling simply becomes a special case of the adaptive multi-round scheduling for systems with complete node homogeneity.

3.2.2 *Adapting the task size*

The initial calculation of F, the fitness index, abstracts ab-initio the resource availability in a given system, but its temporal validity is not necessarily assured as the load conditions frequently vary over time.

In our adaptive approach to multi-round scheduling, we propose to adapt F periodically according to the latest performance reading for a node.

Let us examine an illustrative case involving four workers, w_1, w_2, w_3, and w_4, with calibration times of 1, 2, 3, and 4 time units, respectively. Suppose that initially $S = 68$ and bear in mind that the first four elements are processed during calibration. Thus,

$$F_1 = 0.48, F_2 = 0.24, F_3 = 0.16, \text{ and } F_4 = 0.12$$

by (4)

$$\bar{t} = 2.5, \sigma = 1.3, \text{ and } CV = 0.5 \ (k \approx 2)$$

by (7) and (8).

As per (3), implemented in practical terms as shown in (9), half of the remaining workload ($S = 64/2$) will be initially distributed to the workers w_1, w_2, w_3, and w_4 in chunks of size 15, 8, 5, and 4.

$$\alpha_\iota = \left\lfloor \frac{S}{k} \times F_\iota + 0.5 \right\rfloor \tag{9}$$

Let us suppose that the initial calibration times are preserved as a result of an unchanging node availability, hence the expected execution times for the assigned task sizes will be 15, 16, 15, 16. Given that w_1 reports first for the next installment, the farmer will then assign 8 elements as now $S = 32$. Then, if w_3 follows, the installment will be 2 as now $S = 24$ after the assignation to the first worker. The full installment sequence for each worker and the timing chart are presented in Table 1 and Fig. 3, respectively.

Note that the resulting installment sequence chiefly follows a geometrical progression with ratio $1/k$ and reflects the load balancing spirit of the algorithm. Furthermore, as the task sizes $\alpha_1 = 32, \alpha_2 = 15$, $\alpha_3 = 10, \alpha_4 = 7$ ponder the fitness of every worker, so does the number of installments per node 8, 5, 5, 4. The combination of these characteristics intrinsically reduces the possibility of load imbalances, as larger chunks are initially assigned to reduce scheduling overhead, then smaller chunks are distributed, and, at the end, their size is always one, reducing the load imbalance while maintaining resource awareness.

The aforementioned conditions hold true if and only if the fitness of every node remains constant over the execution of the workload. However, one of the principal characteristics of non-dedicated heterogeneous systems is their dynamism. Let us assume that the w_4 performance/availability doubles during the execution of its first chunk composed of 4 elements, resulting in an execution time of 8 instead of the expected 16. As per (4), this modifies its own and the

Table 1 Actual installment sizes for each worker, e.g. the fourth worker, $w4$, has a sequence of installments of size 4, 1, 1, 1 (or a task size of $\alpha_4 = 7$) with corresponding durations of 16, 4, 4, 4

Installment	α_1	α_2	α_3	α_4
1	15	8	5	4
2	8	3	2	1
3	3	2	1	1
4	2	1	1	1
5	1	1	1	
6	1			
7	1			
8	1			
Total (α_i)	32	15	10	7
Termination time	32	30	30	28

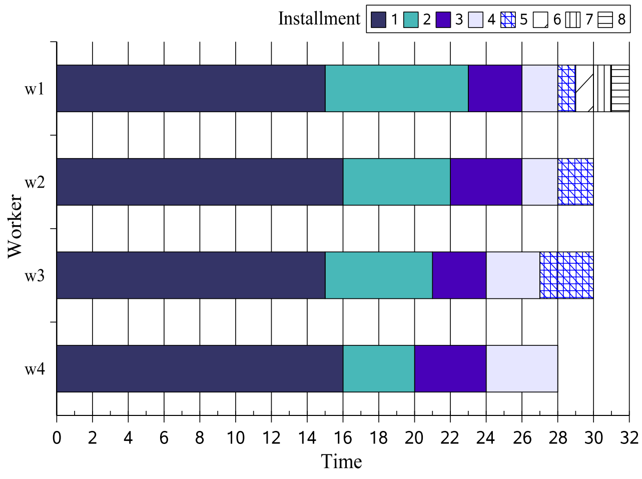


Fig. 3 (Color online) Graphical representation of the installment timing sequence for each of the four participating workers $w1$, $w2$, $w3$, and $w4$, taking into account their associated calibration times of 1, 2, 3, and 4 time units respectively

other nodes' fitness as ($F_1 = 0.43$, $F_2 = 0.215$, $F_3 = 0.14$, $F_4 = 0.215$) and, consequently, the installment sizes, e.g., the next installment for w_4 is 3.

As a result of this feedback through the latest execution time for each node, the fitness index value is constantly refreshed for each processor. Nonetheless, it is also important to emphasise that the summation of the fitness indices (F_i) is always equal to one, as initially defined in (1), regardless of the number of processors and the value of the installment factor.

It should be clear that, by recalculating the fitness of every node according to its latest execution time, the execution phase assimilates immediate feedback not only to the node but also to the system as a whole. As the performance of a node is mainly defined by its system load, this technique arguably adapts the TF execution according to the prevailing load conditions, which can be conceived as the equivalent to a continual system-wide calibration.

Figure 4 illustrates a realistic 8-worker example in a non-dedicated heterogeneous cluster using $S = 9600$ and $k = 2.735$. Each chart depicts the installment sequence as a continual line, where the size of every installment is indexed to the left y-axis and correlated with the prevailing load conditions indicated with the dashed bars indexed to the right y-axis. The system load value is the 1-minute node/worker load average as displayed by the Linux `uptime` command.

Thus, chart (b) represents the 7-installment sequence for w_2 with sizes 454, 263, 161, 159, 2, 1, and 1 under loading conditions of 0.94, 2.3, 1.14, 0.96, 7.47, and 7.23. Note the dramatic reduction between the fourth and the fifth installments as a result of the 7-fold load increase, or the nearly-constant size between the third and the fourth installments as a result of the load reduction. Chart (f) has a more linear behaviour, as the w_6 load follows a more steady pattern.

Although it is difficult to accurately characterise the entire system and the algorithm behaviour, the eight charts provide a succinct illustration of the overall functionality.

Although k is dependent on the calibration times of the nodes and can arguably be modified every time the fitness is affected, we have decided not to recalculate it every time to avoid overhead, as it only serves as a geometric ratio in the progression, rather than a determining factor for feedback.

4 Implementation

For convenience, we have implemented a simple algorithmic skeleton for the task farm.

Algorithmic skeletons abstract commonly-used patterns of parallel computation, communication, and interaction (Cole 2004, 1989). While computation constructs manage logic, arithmetic, and control flow operations, communication and interaction primitives coordinate inter- and intra-process data exchange, process creation, and synchronisation. Skeletons provide top-down design, composition, and control inheritance throughout the program structure. Parallel programs are expressed by interweaving parameterised skeletons analogously to the way in which sequential structured programs are constructed.

In order to use our implementation, one needs to define the tuple $\langle I, O, f \rangle$—where I and O are the input and output vectors, and f is the worker function—and the scheduling mode. It requires no further input from the user. Based on the prevalent load conditions of the defined platform, the calibration phase then automatically calculates the F and the corresponding number of tasks per node α_ι and proceeds according to the selected TF scheduling.

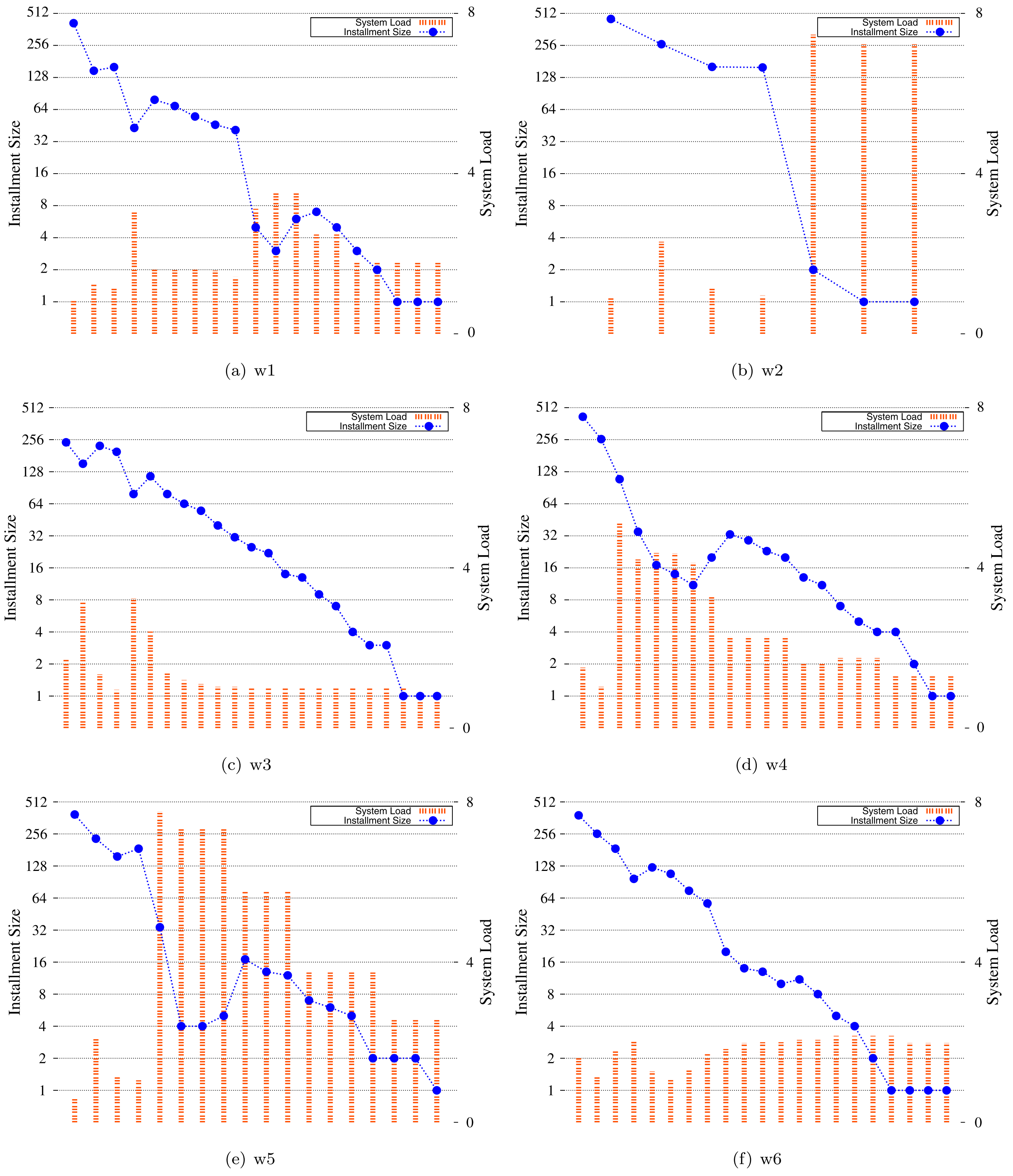
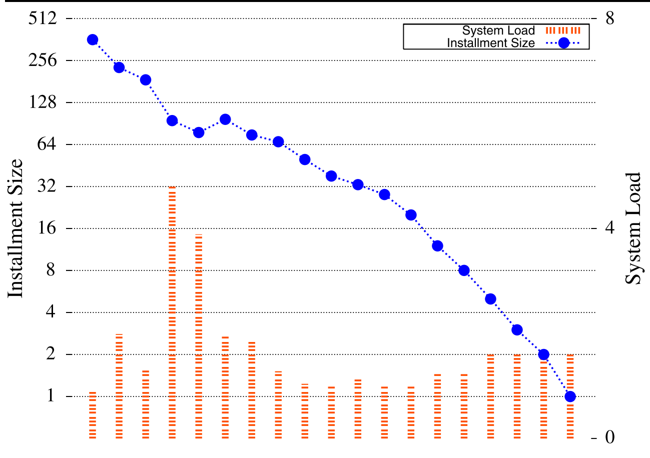
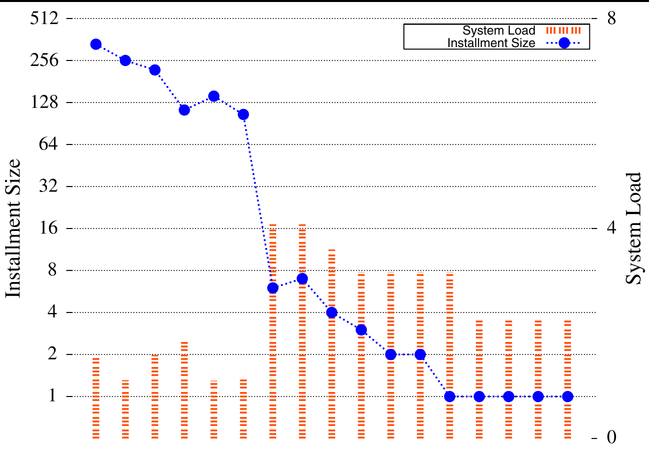


Fig. 4 (Color online) An empirical example of the functionality of the task farm adaptiveness on an actual 8-worker system. For each worker, $w1$ to $w8$, the chart depicts with a solid line the installment size sequence with its value indexed to the left y-axis, and, with dashed bars, the system load present at the node when that given installment is distributed with its value indexed to the right y-axis

Figure 5 presents the algorithmic skeleton API implementing the TF. We stress that it is merely a syntactic vehicle to support the investigation of application scheduling schemes, which forms our main contribution. A more so-



(g) w7



(h) w8

Fig. 4 (*Continued*)

Table 2 The six different scheduling modes for our task farm skeleton

No.	Name	Values	Description
1	SCH_TRAD	$\alpha_t = 1$	Traditional multi-round scheduling based on a work queue (1 by 1)
2	SCH_DEAL	Equation (3) holds. ($k = 1 \wedge F_t = \frac{1}{N}$)	Single-round scheduling assuming equal task sizes for the N nodes
3	SCH_DEALDYN_LR	Equations (3) and (4) hold. ($k = 1$)	Single-round scheduling with statistical univariate calibration (t adjusted via curve-fitting)
4	SCH_DEALDYN_MV	Equations (3) and (4) hold. ($k = 1$)	Single-round scheduling with statistical multivariate calibration (t adjusted via curve-fitting)
5	SCH_DEALDYN_SM	Equations (3) and (4) hold. ($k = 1$)	Single-round scheduling with times-only calibration
6	SCH_MULTI	Equations (3) and (4) hold. ($k = \ln(S)^{CV}$)	Generic variable chunk-size multi-round scheduling with times-only calibration, and single-round as special case ($CV \simeq 0$)

phisticated interface could be defined for production use. The API provides sufficient flexibility to accommodate different options in terms of the worker function (`worker`); the type and size of the input (`in_data`, `in_length`, and `in_type`) and output (`out_data`, `out_length`, and `out_type`); the MPI communicator (`comm`); and the scheduling mode (`sched`). In particular, the valid scheduling modes are presented in Table 2.

That is to say, this skeleton can be used unaltered with single-round scheduling either simply (`DEAL`) or with resource awareness (`DEALDYN`) in its three variants, and with multi-round scheduling either non-adaptively (`TRAD`) or adaptively (`MULTI`).

Note that the adaptive mode (`MULTI`) effectively generalises the scheduling of divisible workloads, as single-round scheduling effectively becomes a special case of the multi-round scheduling for systems with low dispersion.

Our current TF implementation employs the GNU Scientific Library (Galassi et al. 2005) to calculate the regression in the statistical calibration and the coefficient of variability in the adaptive multi-round scheduling. The Network Weather Service (Wolski et al. 1999) is used for the forecasts of processor availability (a) and latency (ℓ) in the statistical calibration. Nonetheless, it is important to emphasise that the implementation is open to the use of any other statistical or resource monitoring routines.

5 Experimental evaluation

Our experiments have been designed to take advantage of the TF intrinsic task parallelism—which presents virtually no inter-process communication—and the ability to access different data sources—inherent to any heterogeneous distributed system. As a result, they deploy a parameter-sweep for a series of independent executions of a stochastic sim-

```
void pfarm(void(*worker)(), void *in_data, int in_length, MPI_Datatype in_type, void *out_data,
          int out_length, MPI_Datatype out_type, MPI_Comm comm, enum scheduling sched);
```

Fig. 5 The application program interface (API) to our adaptive task farm algorithmic skeleton

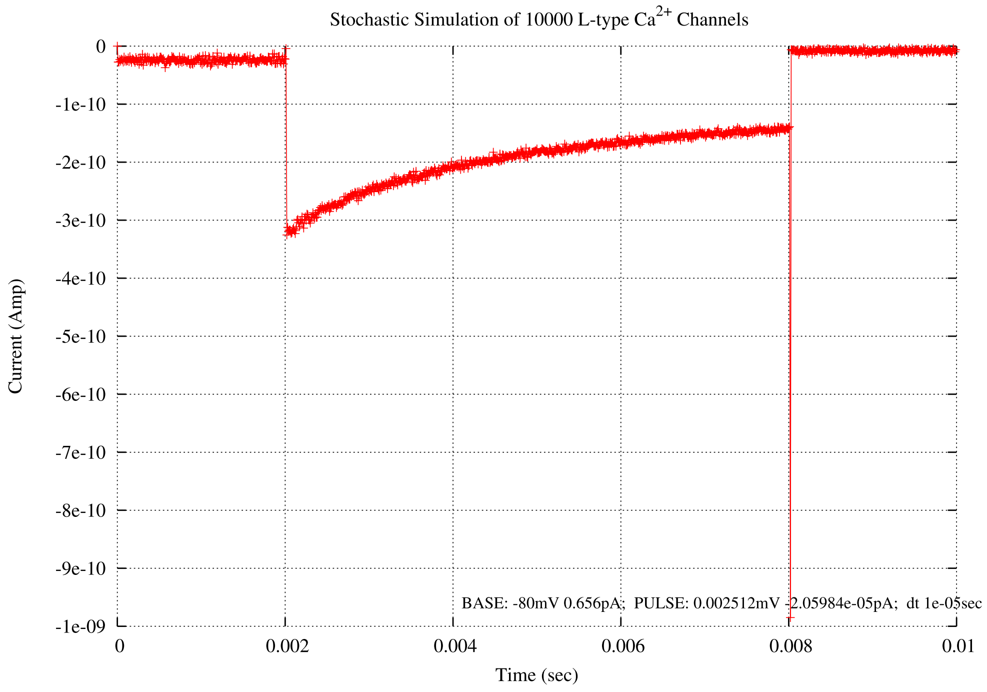


Fig. 6 (Color online) A calcium concentration graph generated by an illustrative run of the parameter sweep, employing 10^4 channels and simulation time 10 ms in intervals of 10 µs

ulation algorithm of voltage-gated calcium channels on the membrane of a spherical cell. Parameterised in terms of the number of channels and time resolution, the algorithm calculates the calcium current and generates a calcium concentration graph per run.

A spherical cell possesses thousands of voltage-gated channels, and simulating their stochastic behaviour implies the processing of a large number of random elements with different parametric conditions. Such parameters describe the associated currents, the calcium concentrations, the base and peak depolarising voltages, and the time resolution of the experiment. This process can be modelled stochastically, defining a threshold based on voltage and time constraints, and aggregating individual calcium currents for a given channel population (González-Vélez and González-Vélez 2005).

Furthermore, as the voltages and the peak duration can be varied without affecting the complexity, the parameter space can be explored while preserving the complexity constant at each run. The model has been abstracted as the function f where the number of channels (*channels*) and time resolution, defined as the number of steps (*steps*), determine its temporal complexity on a per-experiment basis as shown in (10).

$$Time(model) = Order(channels \times steps) \tag{10}$$

Thus, a typical experiment involving the simulation of 10^4 channels for a second in 10 µs intervals (10^5 steps) will have *Order* (10^9) temporal complexity. Each experiment generates two result files: a data file which records the calcium currents values over time and a gnuplot script to automatically produce graphs for these values.

Figure 6 presents a typical processed calcium concentration graph for 10^4 channels and a 10-ms simulation time with a time interval of 10 µs, i.e., a time resolution of 10^3 steps and a complexity *Order* (10^7).

The physiological interpretation of the algorithm is beyond the scope of this work, nonetheless it is interesting to underscore its relevance to the biomedical community. A complete description of the simulation algorithm, a comprehensive parameter sweep, and the physiological interpretation of the results are reported by González-Vélez and González-Vélez (2007).

In the following sections we present a series of experiments which explore the parameter space in breadth and width: the single-round ones cover statistical and times-only calibration for a single problem size (breadth), while the

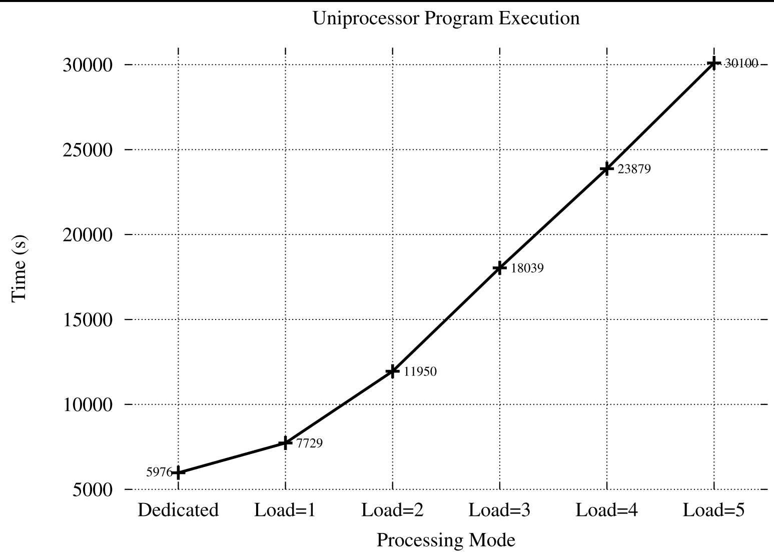


Fig. 7 Uni-processor execution of the workload under variable load conditions. It employs a sequential version of the worker function in a single processor, and load-generating function. In the x-axis, the values represent the number of instances of the load-generating program, and the y-axis indicates the execution time in seconds

Table 3 Parameter space for the single-round scheduling task farm. Key: **E**: Experiment; **S**: Sweep

	Parameter	Value
E	Number of Channels	10^4
	Time Resolution	10^4
	Peak Duration	0.06 s
S	Peak Voltage Steps	0.125 mV
	No. of Experiments	960

multi-round focus on times-only for different problem sizes (width).

5.1 Single-round scheduling

For this case study, we have instantiated the parameter space with 960 experiments of similar complexity, $S = 960$, by varying the peak voltage, and have defined O to store the individual times for each experiment. Previously discussed in our work on single-round scheduling (González-Vélez 2006), the full instantiation is shown in Table 3, using a simulation time of 0.1 s with an interval of 10 µs, and the peak voltage varied in 0.125-mV steps.

Initially and as a sanity check, we have implemented the sequential version of the workload, executed it in a dedicated reference node, and observed its performance under increasing load conditions. Figure 7 plots the execution times in seconds under increasing load conditions. The values in the x-axis represent the number of instances of the load-generating program, which is equivalent to 1 in the 1-minute reading from the Linux/Unix `uptime` command. As expected, it degrades linearly when the system load is increased.

Table 4 presents the execution times of a simple TF version on a 1-farmer 1-worker dedicated configuration and compares them to the uni-processor version. The MPI version with single-round scheduling, where the farmer assigns the 960 elements at once to the worker, performs roughly on a par with its uni-processor counterpart (5890 s versus 5976 or <2% difference). The MPI version with work queue scheduling, where the farmer assigns a single task at once, is 23% slower than the uni-processor version (7330 s versus 5976 s), and this is mainly due to the overhead incurred by the frequent communication. All entries represent the arithmetic mean, with a small variance, of a series of executions.

For our main evaluation, we have deployed three variants of the single-round scheduling: times-only and linear regression in univariate and multi-variate modes. For the univariate case, we have used the scaled availability, a', as predictor variable, and for the multi-variate, we have additionally employed the network latency, ℓ. Both fit the execution times t using linear regression. The baseline is an isometric-installment single-round scheduling, i.e., equal task sizes to all participating workers.

We have run a series of experiments with 6, 12, 24, and 48 worker processes mapped to an equal number of nodes with the farmer located at process 0. The results are presented in Fig. 8 and are based on the aforementioned 960-experiment parameter sweep.

We have chosen the BogoMips (van Dorst 1996) as the known benchmark value in order to scale the availability values, based on its wide availability in Linux systems and its claims to reflect more accurately the processing power of a node than the standard CPU frequency. Nonetheless, the API is not tied to this benchmark, and, alternatively, our evaluation can potentially use the 1-processor figures from any other widely used benchmark.

Each value in the chart represents the average of the executions run under different conditions on three different days. All times are measured at system level and include not

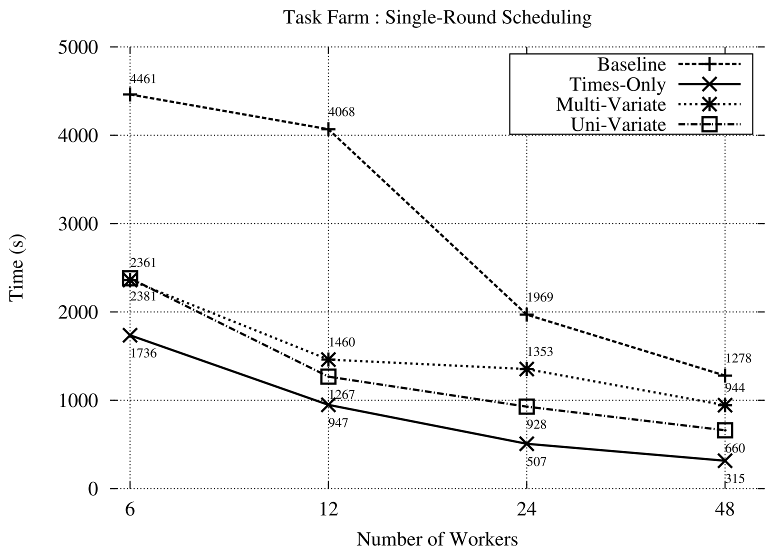


Fig. 8 Summary of the execution times, in seconds, of the task farm with single-round scheduling using 6, 12, 24, and 48 workers. Key: [Baseline] Isometric-installment single-round scheduling; [Times-Only] Single-round scheduling with times-only calibration; [Multivariate] Single-round scheduling with statistical calibration using a' and ℓ as predictors; [Univariate] Single-round scheduling with statistical calibration using a' as predictor

Table 4 Execution times, in seconds, of the task farm skeleton for the 960-experiment parameter-sweep. Cases No. 1 and 2 are the MPI version using one farmer and one worker, with single-round and work queue scheduling respectively. Case No. 3 is a sequential version employed as the baseline. The three cases represent the average of five executions on a dedicated system

Case No.	Experimental version	Execution time
1	MPI single-round	5890 s
2	MPI work queue	7330 s
3	Baseline (uni-processor)	5976 s

only the TF processing but also the calibration and startup-termination periods. The experiments did not run concurrently, in order to avoid any contention. During three different days, the evaluation series coped with different system loads and network conditions.

Our single-round scheduling evaluation consistently outperforms the single-round version using isometric installments by 70% for the times-only calibration and 56% and 48% respectively for the univariate and multivariate linear regression cases. Furthermore, if we compare the different modalities of our single-installment scheduling, the times-only calibration performs 43% and 33% better than the statistical calibration modes. Such performance superiority can be possibly associated with the arithmetic operations generated by the linear regression. Thence, we intend to use times-only calibration for the remainder of our experiments.

5.2 Multi-round scheduling

Here we have run a more comprehensive series of experiments incorporating three different complexities: light, medium, and heavy.

At the single experiment level, while maintaining the number of calcium channels, the time interval, and the base voltage constant at 10000, 10 μs, and −80 mV, respectively, we have varied the simulation time for each experiment using 0.01 s, 0.1 s, and 1 s, i.e., a time resolution of 10^3, 10^4, and 10^5 steps, respectively. Note that the complexity of the experiments in the medium case is similar to that of those employed in the preceding section for the single-round scheduling.

Table 5 Parameter space for the multi-round scheduling task farm. The final time rows show the average execution time, in seconds, for a whole parameter sweep on 8, 16, and 32 workers using adaptive multi-round scheduling. Key: **E**: Experiment; **S**: Sweep

	Parameter	Light	Medium	Heavy
E	No. Channels	10^4	10^4	10^4
	Time Resolution	10^3	10^4	10^5
	Peak Duration	0.006 s	0.06 s	0.6 s
S	Voltage Steps	0.0125 mV	0.03125 mV	0.125 mV
	No. Experiments	9600	3840	960
	8-worker Time	868.4 s	3199.6 s	7720.8 s
	16-worker Time	470.4 s	1730.2 s	4205.1 s
	32-worker Time	235.3 s	869.2 s	2125.1 s

At the parameter space level, the parameter sweep looks upon peak voltages in $[-60, 60\text{ mV}]$. Employing steps of 0.0125 mV, 0.03125 mV, and 0.125 mV, this range is evenly divided, producing an associated number of experiments of $S = 9600$, 3840, and 960, respectively. Note that the variation in the value of S has no bearing on the complexity of the experiments as described in (10). Table 5 shows the three instances of the parameter space.

We have assembled nine different scenarios by varying the number of workers $8, 16, 32$ executing the light, medium, and heavy complexities, and compared our adaptive scheduling with the work queue which is the de-facto

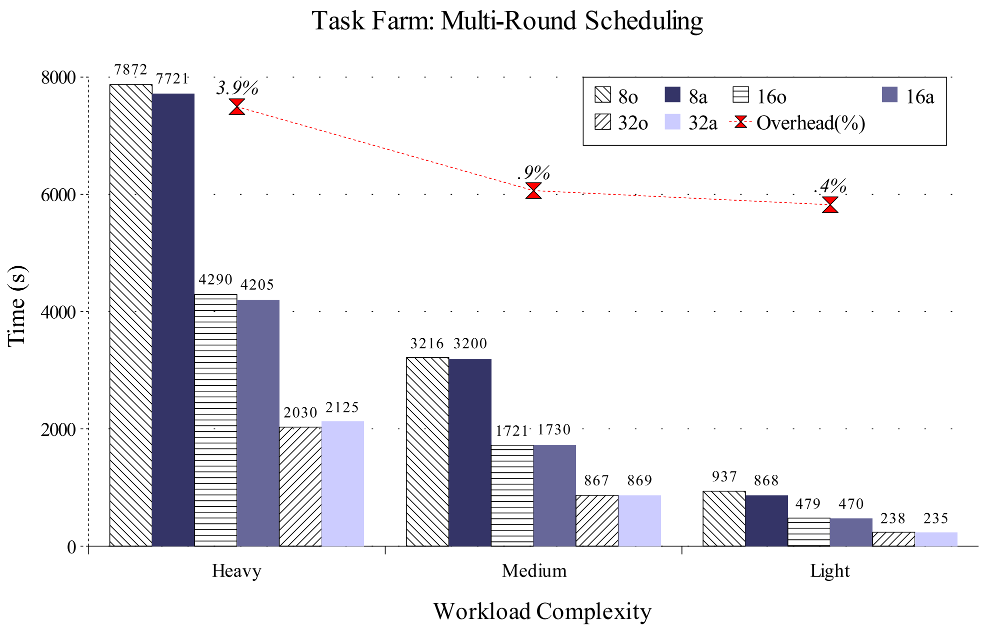


Fig. 9 (Color online) Execution time summary for the task farm using adaptive multi-round scheduling on 8, 16, and 32 workers with light, medium, and heavy problem complexities, as described in Table 5. The *hatched* and *solid bars* represent the execution times for each combination of worker-scheduling-complexity setting for one-by-one and adaptive scheduling modes, respectively. The *top dotted line* represents the overhead incurred by the initial calibration. Key: [processors no.] [scheduling mode], e.g., 8a means 8 workers and adaptive scheduling model and, analogously, 8o represents 8 workers and one-by-one scheduling

scheduling for heterogeneous, dynamic systems. While the results have demonstrated a performance improvement of 8% for the light case with eight processors and more modest gains for the remaining light, medium, and heavy cases, the automatic calculation of the installment factor and the periodic refinement of the task size should be considered important contributions for self-scheduling parallelism. What is more, the overhead incurred varies from 3.9% for the heavy complexity to 0.4% for the light one. A summary of the results is presented in Fig. 9.

6 Conclusions

While simulation and theoretical formulations have traditionally provided a preponderant foundation for generic approaches in the study of scheduling, there is a clear need for empirical work to inquire into the performance of adaptive algorithms.

In this work, we have investigated the feasibility of using a skeletal task farm to schedule divisible workloads and, consequently, enhance the performance of the corresponding parallel programs. Being agnostic to the application itself, our methodology has deployed a pragmatic approach in order to instrument the parallel program at compilation, allowing it to adapt at execution.

By implementing a realistic parameter sweep application, we have evaluated our approach using two different scenarios:

1. a single-round scheduling, which employed times-only and statistical calibrations with no feedback, and
2. a multi-round scheduling with times-only calibration and periodic adaptation throughout the execution. In addition to this continual adaptivity, this case has illustrated the ability to automatically discriminate between the multi- and single-round scheduling by introducing an installment factor based on the dispersion of the calibration times of the participating nodes.

While the suggested formula for the installment factor has helped us to construct a useful DLT heuristic based on the dispersion of the nodes and the number of tasks, further research is required to demonstrate its optimality and/or uniqueness. We strongly believe that as long as the installment factor is correlated to tasks and dispersion, it will provide a useful guidance to steer the number of scheduling rounds.

As proven by the uni-processor figures, the load in the system directly impacts upon the execution times. Hence, it is crucial to note that the adaptive method shows advantages regardless of the system load. In other words, the adaptive farm is able to adjust itself to the dynamism of the environment.

Section 5.1 has provided the evidence that the calibration phase of the worker function on the nodes can considerably enhance the performance of a task farm, and F can help in predicting variations in system conditions.

The corrective properties of linear regression to relieve exogenic factors in larger runs, such as the arrival of administrative jobs or indiscriminate interactive usage, were thought to be useful. However, times-only calibration has proven to be the most effective for our purposes.

From an efficiency perspective, it is arguable that the single-round performance ought to be enhanced by conveying dynamic re-calibration into the distribution when any performance bottlenecks arise. Therefore, we have presented the evaluation of the generic multi-round case which automatically calculates the installment factor based on the dispersion of the calibration times of the nodes, and pervades the impact of changes in the nodes' fitness through a periodic adjustment. Despite their modest performance results, the proposed algorithms have substantial implications for self-scheduling and load-balancing.

With respect to the analysis of divisible workloads, the findings of the case study provide an alternate approach to the single-round scheduling problem using forecasts of resource utilisation. This tacitly reinforces the notion that although heterogeneous systems are often highly dynamic, forecasts based on historical resource utilisation can accurately provide some guidance for distributing workloads.

References

Banino, C., Beaumont, O., Carter, L., Ferrante, J., Legrand, A., & Robert, Y. (2004). Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Transactions on Parallel and Distributed Systems*, *15*(4), 319–330.

Beaumont, O., Legrand, A., & Robert, Y. (2003). Scheduling divisible workloads on heterogeneous platforms. *Parallel Computing*, *29*(9), 1121–1152.

Beaumont, O., Casanova, H., Legrand, A., Robert, Y., & Yang, Y. (2005). Scheduling divisible loads on star and tree networks: Results and open problems. *IEEE Transactions on Parallel and Distributed Systems*, *16*(3), 207–218.

Bharadwaj, V., Ghose, D., Mani, V., & Robertazzi, T. G. (1996). *Scheduling divisible loads in parallel and distributed systems*. Los Alamitos: IEEE Press.

Bharadwaj, V., Ghose, D., & Robertazzi, T. G. (2003). Divisible load theory: a new paradigm for load scheduling in distributed systems. *Cluster Computing*, *6*(1), 7–17.

Blazewicz, J., Drozdowski, M., & Markiewicz, M. (1999). Divisible task scheduling—concept and verification. *Parallel Computing*, *25*(1), 87–98.

Cariño, R., & Banicescu, I. (2008). Dynamic load balancing with adaptive factoring methods in scientific applications. *The Journal of Supercomputing*, *44*(1), 41–63.

Cole, M. (1989). *Algorithmic skeletons: structured management of parallel computation. Research monographs in parallel and distributed computing*. London: Pitman/MIT Press.

Cole, M. (2004). Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. *Parallel Computing*, *30*(3), 389–406.

Comino, N., & Narasimhan, V. (2002). A novel data distribution technique for host-client type parallel applications. *IEEE Transactions on Parallel and Distributed Systems*, *13*(2), 97–110.

Drozdowski, M. (1997). *Selected problems of scheduling tasks in multiprocessor computer systems*. *Monographs: Vol. 321*. Poznan: Poznan University of Technology Press. http://www.cs.put.poznan.pl/mdrozdowski/txt/h.pdf. Last Accessed: 17 Jan 2009.

Drozdowski, M., & Lawenda, M. (2007). Multi-installment divisible load processing in heterogeneous distributed systems. *Concurrency and Computation: Practice and Experience*, *19*(17), 2237–2253.

Drozdowski, M., & Lawenda, M. (2008). Scheduling multiple divisible loads in homogeneous star systems. *Journal of Scheduling*, *11*(5), 347–356.

Galassi, M., Davies, J., Theiler, J., Jungman, B. G. G., Booth, M., & Rossi, F. (2005). Least-squares fitting. In *GNU scientific library reference manual, network theory* (Chap. 36, pp. 361–369). Bristol.

Ghose, D., Kim, Hj., & Kim, Th. (2005). Adaptive divisible load scheduling strategies for workstation clusters with unknown network resources. *IEEE Transactions on Parallel and Distributed Systems*, *16*(10), 897–907.

González-Vélez, H. (2005). An adaptive skeletal task farm for grids. In *Lecture notes in computer science: Vol. 3648*. *Euro-Par 2005* (pp. 401–410). Berlin: Springer.

González-Vélez, H. (2006). Self-adaptive skeletal task farm for computational grids. *Parallel Computing*, *32*(7–8), 479–490.

González-Vélez, V., & González-Vélez, H. (2005). A grid-based stochastic simulation of unitary and membrane Ca^{2+} currents in spherical cells. In *CBMS'05*, IEEE, Dublin (pp. 171–176).

González-Vélez, V., & González-Vélez, H. (2007). Parallel stochastic simulation of macroscopic calcium currents. *Journal of Bioinformatics and Computational Biology*, *5*(3), 755–772.

Hagerup, T. (1997). Allocating independent tasks to parallel processors: an experimental study. *Journal of Parallel Distributed Computing*, *47*(2), 185–197.

Kruskal, C. P., & Weiss, A. (1985). Allocating independent subtasks on parallel processors. *IEEE Transactions on Software Engineering*, *11*(10), 1001–1016.

Legrand, A., A, Su., & Vivien, F. (2008). Minimizing the stretch when scheduling flows of divisible requests. *Journal of Scheduling*, *11*(5), 381–404.

Li, P., Veeravalli, B., & Kassim, A. (2005). Design and implementation of parallel video, encoding strategies using divisible load analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, *15*(9), 1098–1112.

Liu, J., Saletore, V., & Lewis, T. (1994). Safe self-scheduling: a parallel loop scheduling scheme for shared-memory multiprocessors. *International Journal of Parallel Programming*, *22*(6), 589–616.

Martens, H., & Naes, T. (1989). *Multivariate calibration*. Chichester: Wiley.

Polychronopoulos, C. D., & Kuck, D. J. (1987). Guided self-scheduling: a practical scheduling scheme for parallel supercomputers. *IEEE Transactions on Computers*, *36*(12), 1425–1439.

Robertazzi, T. G. (2003). Ten reasons to use divisible load theory. *Computer*, *36*(5), 63–68.

Robertazzi, T. G. (2008). Divisible load scheduling. Web site. http://www.ece.sunysb.edu/~tom/dlt.html. Updated: 18 Oct 2008. Last Accessed: 17 Jan 2009.

van der Raadt, K., Yang, Y., & Casanova, H. (2005). Practical divisible load scheduling on grid platforms with APST-DV. In *IPDPS'05*, Denver, p. 29b.

van Dorst, W. (1996). The quintessential Linux benchmark: all about the "BogoMips" number displayed when Linux boots. *Linux Journal* (21es), 4.

Viswanathan, S., Veeravalli, B., & Robertazzi, T. G. (2007). Resource-aware distributed scheduling strategies for large-scale computational cluster/grid systems. *IEEE Transactions on Parallel and Distributed Systems*, *18*(10), 1450–1461.

Wolski, R., Spring, N., & Hayes, J. (1999). The Network Weather Service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, *15*(5–6), 757–768.

Yang, Y., van der Raadt, K., & Casanova, H. (2005). Multiround algorithms for scheduling divisible loads. *IEEE Transactions on Parallel and Distributed Systems*, *16*(11), 1092–1102.

Yang, Y., Casanova, H., Drozdowski, M., Lawenda, M., & Legrand, A. (2007). *On the complexity of multi-round divisible load scheduling* (Tech. Report 6096). INRIA, ISSN: 0249-6339.