J Grid Computing (2016) 14:153–170
DOI 10.1007/s10723-015-9345-8

Time and Energy Performance of Parallel Systems with Hierarchical Memory

Jędrzej M. Marszałkowski · Maciej Drozdowski · Jakub Marszałkowski

Received: 30 November 2014 / Accepted: 6 August 2015 / Published online: 9 September 2015

Abstract In this paper we analyze the impact of memory hierarchies on time-energy trade-off in parallel computations. Contemporary computing systems have deep memory hierarchies with significantly different speeds and power consumptions. This results in nonlinear phenomena in the processing time and energy usage emerging when the size of the computation is growing. In this paper the nonlinear dependence of the time and energy on the size of the solved problem is formalized and verified using measurements in practical computer systems. Then it is applied to formulate a problem of minimum time and minimum energy scheduling parallel processing of divisible loads. Divisible load theory is a scheduling and performance model of data-parallel applications. Mathematical programming is exploited to solve the scheduling problem. A trade-off between energy and schedule length is analyzed and again nonlinear relationships between these two criteria are observed. Further performance analysis reveals that energy consumption and schedule length are ruled by a complex interplay between the costs and speeds of on-core and out-of-core computations, communication delays, and activating new machines.

Keywords Time-energy trade-off · Divisible loads · Parallel computations · Hierarchical memory · Out-of-core processing

J. M. Marszałkowski · M. Drozdowski · J. Marszałkowski (✉)
Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań, Poland
e-mail: jakub.marszalkowski@cs.put.poznan.pl

J. M. Marszałkowski
e-mail: jedrzej.marszalkowski@cs.put.poznan.pl

M. Drozdowski
e-mail: maciej.drozdowski@cs.put.poznan.pl

1 Introduction

Moving end-user information services, scientific computations, data storage and processing to *the cloud* is a very active trend in the current information technology. Huge datacenters hosting thousands of servers are the driving force behind the cloud metaphor. Similarly to the heavenly clouds which have considerable size and weight, the compute-cloud has non-negligible cost and environmental footprint. Datacenter energy consumption is a key determinant of the cost of running the compute-cloud. Thus, limiting datacenter energy consumption is a must to curb operational costs and to meet the requirements of environmentally-friendly computing. Furthermore, power grids have their limitations which restrict the further growth of the datacenters [16, 21, 22]. Not only 'big iron' datacenters are affected by energy limitations. Energy consumption is a significant aspect also in the lightweight end of the computer systems spectrum: in mobile devices, sensor networks, and in aerospace applications.

Energy savings may be obtained by commissioning new hardware, refactoring software, but also by effective planning and scheduling of the computations. This results in increasing interest in energy-efficient scheduling of machines at the grid level [10, 31, 33]. Proper planning is especially important in the distributed case because communications and computations must be coordinated to avoid wasting energy in idle machines. What is more, some machines may be in energy-saving modes, such as suspension to disk (hibernation), so it is costly to start them. Situation becomes even more complicated if we take into account the fact that computer systems have deep memory hierarchies with significant differences in speed and power consumption. Then, we have a number of design dilemmas and questions. For example, should the work be done on a given set of machines with the higher cost of using slower memory, or on a larger number of machines at the cost of powering them on? What is the trade-off between energy and schedule length? How does this trade-off change if the number of machines, problem size, speed of storage change? Note that energy and time of computation, communication, machine activation cannot be considered separately because savings achieved in one dimension of scheduling may expose deficiencies or bottlenecks elsewhere [7].

In this paper we analyze scheduling parallel computations in systems with hierarchical memory. Although memory hierarchy is present in almost all current computing systems, it is rarely taken into account in scheduling or energy considerations. We assume two memory levels: for on-core and for out-of-core computations. Each memory level has different speed and electric power consumption which results in nonlinear dependencies between computation time, energy consumption and memory usage. The dependencies are formalized and empirically validated. Then a problem of scheduling computations on a distributed system with hierarchical memory is formulated. Time and energy costs of communication and machine activation are taken into account. Parallel computations are represented as divisible loads. Divisible load theory (DLT) [6, 12, 17] has been developed to schedule and analyze performance of data-parallel (i.e. Big Data) computations in distributed systems. Thus, the methodology of DLT is well suited to represent, e.g., cloud-oriented batch processing operations. DLT is further introduced in Section 2. Note that in our problem we have two criteria: schedule length (makespan) and energy which makes the problem effectively bicriterial. It can be expected that a trade-off between these two criteria may arise and schedule length may be shortened at the cost of higher energy consumption. The scheduling problem is formulated as a mathematical program minimizing one criterion subject to a limit on the other criterion. In order to address the questions risen above we evaluate by simulation the changes in the time-energy trade-off under varying problem size, system size, speeds and power usage. The key contributions of this paper can be summarized as follows:

- A new model of energy-aware parallel computation with memory hierarchy suitable for the compute-cloud is proposed.
- The assumptions in the model are verified experimentally.
- Time-energy performance trade-off under varying system and application parameters is studied to discover existing relationships and bottlenecks.

The organization of the paper is the following. In the next section we discuss related research on the considered subject. In Section 3 we introduce a mathematical model of the time and energy dependence on the computation size, and report on the experiments conducted to verify this model. The problem of scheduling divisible computations under energy limitations is formulated in Section 4. Section 5 is dedicated to a detailed delineation of energy and time relationships arising as a result of interactions between on-core, out-of-core computations, communications, idle waiting and machine activation costs. In Section 6 we give a broader view of the impact of the system and application parameters on time and energy performance of parallel computations. The final conclusions are drawn in Section 7. The notation used through the paper is summarized in Table 2.

2 Related Work

In this section we report on issues immediately related to our study: hierarchical memory performance, out-of-core computations, and divisible load theory (DLT).

Almost all contemporary computer systems use hierarchical memory to deal with the disparity of CPU and storage speeds. Memory hierarchy may include the following levels:

1. CPU registers,
2. CPU cache levels L1, L2, L3 and even L4,
3. Random Access Memory a.k.a. core memory (often acting as a cache for lower memory levels),
4. Solid State Drives (also acting as a cache of HDD [18]),
5. Hard Disk Drives (also acting as a cache for remote storage),
6. network storage (e.g. NAS using AoE, FCoE, NFS, SMB, and similar),
7. tape, optical devices and other forms of long-term storage.

When we go down the hierarchy, from CPU registers to the external storage, several things change: Capacity increases while time performance decreases. Usually time performance deteriorates both in terms of latency and throughput. The size per storage device increases and price per storage unit (e.g. GB) decreases while moving down the list. The last two issues play more of a key role when designing datacenters rather than in scheduling parallel applications. Things get more complicated if one includes in this scheme the memory of modern graphics cards [27]. Introducing new forms of fast storage, such as SSD, between RAM and HDD, shifts the balance in time and energy costs between the I/O software stack and hardware. Since the intermediate SSD storage is faster, the I/O operations are called more often thus exposing the computational costs of the I/O software stack [39]. Such complex interactions lead to counterintuitive conclusions (cf. "*Software Considered Harmful*" in [39]). It demonstrates that a deeper study of the impact of changing system parameters on time and energy performance is needed. Memory architecture is a broad research and engineering field certainly exceeding the scope of this paper. Interested readers may find further details, e.g., in [18, 32, 38].

In this paper we focus on data-parallel applications processing volumes of data in the order of GBytes and bigger. Such computations run for minutes and longer. Therefore, instruction level parallelism corresponding with operations at memory levels 1 and 2 does not conform with the level of abstraction considered here. Contrarily, it makes a great difference whether the computations are performed in the core memory (i.e. in RAM) or with the use of the external storage. Hence, in the further considerations we will distinguish just two types of computation: at memory levels 1-3 in the above classification which will be conventionally called *on-core* processing, as opposed to the operations involving also levels 4 and upper which will be conventionally called *out-of-core* processing. It is obviously advantageous to perform computations with data entirely stored in RAM compared to the use of external storage. However, since fast memory levels, such as RAM, are always scarce it is sometimes unavoidable to conduct computations with the use of external storage. Thus, out-of-core computations come in handy: they are slower, but more data can be processed.

Research on out-of-core computations considers specialized algorithms, dedicated data partitioning and data access scheduling, recently also with the use of GPU platforms [2, 36]. One essential out-of-core application is sorting big volumes of data [20, 23, 41]. Furthermore, sorting is a key component of the MapReduce distributed processing paradigm [9, 40]. Thus, MapReduce applications performing numerous operations on external storage in text, measurement, image processing, machine learning, and simulation can be considered, broadly speaking, out-of-core computations [24, 26]. Another type of application typically performing computations on many levels of memory are database management systems. Not surprisingly the technology of MapReduce is becoming a component of the NoSQL databases [4, 11, 30]. Rarely in the context of scheduling and performance tuning in such applications is a balance between the on-core and out-of-core computations searched. A recent trend in cloud computing, DBMSes and MapReduce applications, is to fit all the data in RAM [42]. On the one hand it is a reasonable tendency, on the other hand coercing data and code to fit in RAM may have a big price exceeding the costs of the economical use of external storage. It may also be impractical with large volumes of data. Thus, a second trend can be observed of proposing solutions using RAM as a framework for greater amounts of data stored on lower levels of hierarchy, and doing so transparently from the point of view of the applications [28]. This way computations fluently go

out-of-core. In this paper we explore options for finding an optimum balance in partitioning the data and computations between the two main memory levels and we will show (Section 6) that putting everything in RAM is not always the best option.

Divisible Load Theory (DLT) is a paradigm of scheduling and modeling data-parallel computations. A central assumption of DLT paradigm is that the computation consists in processing big volumes of data (here called load) which can be partitioned into parts of arbitrary sizes to be processed independently in parallel. In other words, it means that grains of data are small enough in relation to the whole data volume to allow almost continuous partitioning of the input volume. Moreover, the grains are mutually independent which allows for parallel processing without precedence constraints. DLT was introduced independently in [3] to schedule distributed parallel computations and in [8] to plan distribution of computations and communications in a network of intelligent sensors. In the following years DLT proliferated in many directions to cover various interconnection topologies, alternative communication and computation scheduling strategies, systems with flat memory, time-dependent machine availability or speed. DLT is a valuable scheduling and performance modeling instrument because its assumptions are detailed enough to incorporate many important features of real systems, but at the same time DLT is general enough to allow both flexible analytical modeling and low complexity solutions. Due to space limitations we direct readers interested in the research on DLT to the surveys [6, 12, 17, 34, 35]. As the DLT model is based on a few simple assumptions, its accuracy has been tested in studies [3, 14, 25]. The difference between the model and reality was in the range of 1 % and better. Recent papers confirm the utility of DLT in cloud computing [1, 19] and MapReduce [5].

Energy performance studies in DLT emerged relatively recently. Energy can be modeled as some form of cost. Scheduling DLT for minimum costs of computation has been analyzed in [29, 37]. Scheduling divisible loads for minimum schedule length and cost in general has been studied as a bicriterial optimization problem in [37]. In order to grasp the relationships guiding energy consumption in divisible computations the connections between system and application parameters in determining energy usage were represented as two-dimensional maps in [13]. With respect to memory hierarchy, [15] is the only paper considering scheduling divisible computations. In this paper we depart from the assumptions of the earlier research. Firstly, we focus on the specific cost of energy, which has several consequences: Machines may wait in several types of energy-saving modes. Waking a machine up incurs costs both in time and in energy. Though startup times are not new in DLT [12], not all papers take them into account. Initial energy costs are a new component in divisible load scheduling. Secondly, we consider hierarchical memory which introduces a nonlinear dependence of time and energy consumption on the size of the processed load. In the following section we present our model in more detail.

3 Timing and Energy Use Model

The size of used data structures impacts the speed of processing and energy consumption. External storage is much slower than RAM and out-of-core processing takes much longer time. Simultaneously, energy use is much greater. There is a simple explanation of this phenomenon. When a computer is woken up and idly waiting, conducting no computations, it is utilizing some amount of idle state power which will be denoted P^I (even as much as 80-100W [31]). In such a situation the utilization of the CPU is close to 0 %. When computing on data that fits entirely in RAM, CPU utilization goes close to 100 % while using running power P^R (more than 140W). If the problem size exceeds the available memory, computations get much slower because the external storage cannot keep up with the transfer rate of RAM. The CPU must wait, and its utilization decreases while power consumption becomes closer to P^I. However, computer systems are not energy-proportional. Although CPU utilization drops from nearly 100 % to nearly 0 %, power consumption does not drop proportionally; it only decreases toward P^I which is far from zero. Although the CPU may be using slightly less power in the out-of-core computation mode the savings are not proportionate to the decrease of computing speed. Consequently, energy consumption significantly grows. A turning point of switching between two levels of power consumption is the size of RAM memory.

To support our assumptions on time and energy consumption in systems with memory hierarchy, we made a series of experiments with several computer

architectures and applications. Using a DW-6090 Power Analyzer we were measuring energy consumption for growing problem sizes up to definitely exceeding RAM memory size. Example results are shown in Fig. 1. Details of the system configuration are shown in Table 1 as the first computer configuration. The shapes of the relationships for the other two machines are the same. Similar dependence of time on load size was reported in [15]. We observed that energy consumption and time can be modeled by linear functions of load sizes: one for loads smaller than available core memory and the second for loads exceeding it. This can be seen in Fig. 1 for systems with 512MB and 1024MB RAM. So for some computer M_i we have relationships for time and energy, respectively:

$$t_i^R = max\{a^0\alpha_i; a\alpha_i + b\} \quad (1)$$

$$E_i^R = max\{k^0\alpha_i; k\alpha_i + l\} \quad (2)$$

where α_i is the size of the load processed by M_i. Let us now discuss the meaning of the parameters in these functions on the example of the computation time equation (1). There are two linear functions in (1) representing proportional growth of the execution time with the increasing problem size α_i. Parameters a^0 and a represent the angle of the lines; a is greater than a^0 and hence its line is steeper. This is because in the area of out-of-core computations, more time is necessary to process the same portion of the load. Parameter a^0 can be called the on-core computing rate (1/throughput, in seconds per megabyte), while a and b together define the out-of-core computing time. The b parameter decides that both lines will cross in the point where problem size exceeds available memory, so that the out-of-core takes over. Consequently, the value of b is negative. The same applies to the energy use model. Parameters k^0 and k (in Joules per megabyte) with l (in Joules) define energy costs. The values of b and l in equations (1), (2) are linked by RAM size of the machine. Precisely, $(a^0 - a)/b = (k^0 - k)/l = 1\rho$, where ρ is the size of RAM available for the data. Hence, the value of k^0 is connected with a^0 as well as k with a. Faster computations mean shorter time and lower energy consumption. A reverse connection not necessarily exists. For example, we could imagine two generations of processors, with the same speed, but the latter being more energy-efficient. Thus, for both of them, the values of a^0 representing speed would be the same; however values of k^0 representing energy usage would differ.

In Table 1 we present example values of a^0, a, k^0, k obtained by linear regression on the measured data presented in Fig. 1. By no means are they any ultimate values, as they change with machine and application. Still we chose different hardware configurations to discuss how these values work. As we see machine 2 is faster than machine 1 both in on-core computations and in out-of-core computations, because lower a^0 or a mean shorter processing time. However, this comes at a cost and machine 2 is less energy-efficient in out-of-core computations: bigger k means more energy used. This is a result of generally higher power usage of this machine, since waiting for HDD in out-of-core computations is costly. For a comparison at position 3 we present faster disk storage; here HDD was replaced by SDD. The improvement in speed and energy efficiency of out-of-core computations is easily noticeable. As always this comes at some cost. With SDD it

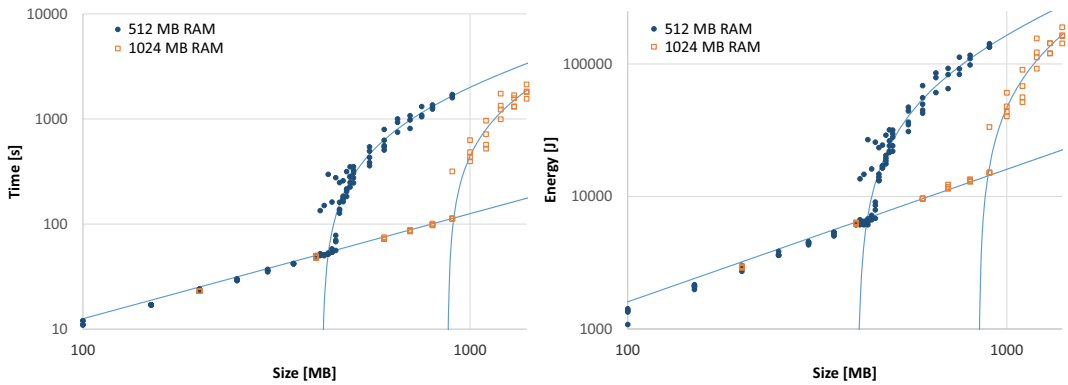

Fig. 1 Time and energy vs problem size measurement (log scale)

Table 1 Computer parameters observed in practice

	a^0 [s/MB]	a [s/MB]	k^0 [J/MB]	k [J/MB]
1. Intel Pentium IV 2.8GHz, HDD Caviar WD400, 512/1024MB DDR 266MHz CL2.5, FreeBSD 9.0	0.126	3.42	16.03	276.5
2. AMD Phenom2 X4 945 3GHz, HDD Samsung 252HJ, 4GB DDR2 800MHZ CL5, FreeBSD 9.0	0.070	2.49	11.00	327.3
3. AMD Phenom2 X4 945 3GHz, Kingston SSDNOW300, 4GB DDR2 800MHZ CL5, FreeBSD 9.0	0.070	0.40	11.00	73.6

is the cost of equipment and its faster wearing off. The values of b, l can be calculated from the data in Table 1 by the fact that the out-of-core and on-core lines cross at ρ which is the size of RAM available for data. Thus, $b = \rho(a^0 - a)$ and $l = \rho(k^0 - k)$.

4 Mathematical Model and Solution Procedure

In this section we formulate a problem of time- and energy-efficient scheduling of divisible loads in systems with hierarchical memory. The notation used through the paper is summarized in Table 2. We assume that there is a load of size V to be processed on m homogeneous machines. We will be looking for the shortest processing time T subject to the lowest possible energy E.

The time schedule of communications and computations is shown in Fig. 2a. The process starts with the load located at the originator (initiator, resource allocator, etc.), a special computer in the network further denoted as M_0. The originator is connected with all slave machines (computers, processors) $M_1, \ldots, M_m$, by means of some network with communication rate C. The originator is dividing and distributing the load; communications are performed only between M_0 and the slaves, one at a time. This way, the load of volume V in chunks of size $\alpha_1, \ldots, \alpha_m$ is sent to machines $M_1, \ldots, M_m$, respectively. Machines take nonzero startup time t^S before they become capable of performing communication. This might represent a simple waking up time, as well as more complicated processes like loading appropriate VMs or platforms in dynamically scalable clouds. Startup time t^S is an

Table 2 Summary of notation

α_i	load assigned to machine i [MB]
a^0	processing rate on-core [s/MB]
a, b	parameters of computation time out-of-core [s/MB]
C	communication rate (1/bandwidth) [s/MB]
E	schedule energy [J]
k^0	energy rate per data unit on-core [J/MB]
k, l	parameters of energy consumed per data unit out-of-core [J/MB]
m	number of machines
ρ	size of RAM available to store data
t_i^I, P^I, E^I	idle time [s], power [W] and energy [J]
t^S, P^S, E^S	start-up time [s], power [W] and energy [J]
t_i^N, P^N, E^N	networking time [s], power [W] and energy [J]
t_i^R, P^R, E^R	running time [s], power [W] and energy [J]
T	schedule length [s]
x_i	decision variable =1 if computer i is activated; =0 otherwise
V	size of load to process [MB]

important element of the DLT model, because without it, an arbitrary number of processors may be activated, which is unrealistic [6, 12]. When the transfer of a chunk of load to machine M_i is finished, M_i starts processing it, while the originator activates machine M_{i+1} in order to send load α_{i+1} to it. The procedure is repeated until starting computations on all m processors.

It is often assumed in DLT that the time of returning results is negligible. With such an assumption it can be shown [6, 8, 34] that in a schedule of the optimal length all machines finish computations at the time T. This is often called an *optimality criterion* in DLT. The result collection procedure can be included in the model, for example, by an appropriate increase of the communication time when machines receive their chunks of load. Discussion on extensions of the DLT networking model including results collection time, parallel communications or communications concurrent with computations can be found in [6, 8, 12, 34]. For intelligibility of the further analysis such alternative communication and computation strategies are not considered.

It is assumed that a machine can be in one of four states: idle (I), starting up (S), networking (N) or running, i.e. performing computations (R). With these states power consumption rates P^I, P^S, P^N, P^R and durations t_i^I, t^S, t_i^N, t_i^R are connected, respectively. As these states might be of complicated nature, we assume that the powers are averages well representing the on-going processes, so that the product of power consumption and time gives overall energy usage $E^X = t^X P^X$, where X represents one of the states: I, S, N, R. Startup time t^S is the time a machine needs to wake up from idle and become operational, i.e. start networking or computations. The value of t^S is equal for all machines. What is called idle here can represent various states in reality. Firstly, the machine can be turned off or hibernated to HDD and waiting for a signal to boot up. The corresponding P^I value will be the lowest, possibly a few Watts, but the startup time will be the longest one, even up to dozens of seconds depending on the software to be loaded. Secondly, the machine can be suspended to RAM, then the startup time will be at the level of seconds, but the power consumption will be around several dozens of Watts. Finally, the machine can be on and waiting to start operating on a new task within a second, but the idle power rate will be up to 100W.

The energy consumed by machine M_i can be calculated as:

$$E_i = E_i^S + E_i^I + E_i^N + E_i^R$$

The running energy E_i^R depends on the size of assigned load α_i as determined by equation (2). As the communication rate is C, the total communication time is calculated as $t_i^N = C\alpha_i$ and energy as $E_i^N = P^N C\alpha_i$. If some machines are not used in the schedule, their loads are $\alpha_i = 0$ so in effect $E_i^N = 0$ and $E_i^R = 0$. However, we need a binary decision variable x_i indicating whether machine i is used in the schedule. Thus, the energy consumed in the startup is $E_i^S = x_i t^S P^S$. The idle time can be calculated from the length of the schedule T and the remaining three times $t_i^I = T - t^S x_i - C\alpha_i - t_i^R$. Hence, we get:

$$E_i = x_i t^S P^S + t_i^I P^I + C\alpha_i P^N + \max\{k^0\alpha_i, k\alpha_i + l\} \tag{3}$$

Energy E_0 consumed by the originator M_0 is calculated differently. It uses power P^N when other machines are starting up or during communication and the originator goes idle when all the load is distributed. The originator cannot go idle when some other machine is waking up, because this would require some wake up time from him too. Including this into the model complicates it beyond any precision gained with such an addition. Thus, we get:

$$\begin{aligned} E_0 = {} & P^N \left(\sum_{i=1}^{m} x_i t^S + \sum_{i=1}^{m} t_i^N \right) \\ & + P^I \left(T - \sum_{i=1}^{m} t_i^N - \left(\sum_{i=1}^{m} x_i t^S \right) \right) \end{aligned} \tag{4}$$

Considering that

$$\sum_{i=1}^{m} t_i^N = C \sum_{i=1}^{m} \alpha_i = CV$$

equation (4) can be transformed to a more convenient form:

$$E_0 = P^N \left(\sum_{i=1}^{m} x_i t^S + CV \right) - P^I \left(\sum_{i=1}^{m} x_i t^S + CV \right) + P^I T \tag{5}$$

$$= P^I T + \left(\sum_{i=1}^{m} x_i t^S + CV \right) (P^N - P^I) \tag{6}$$

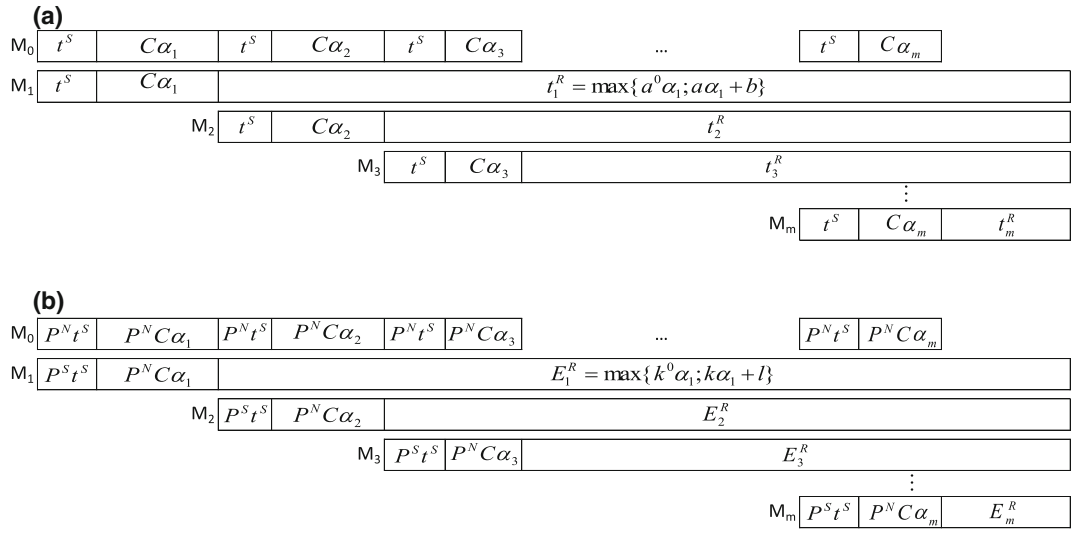

Fig. 2 Model of the DLT computation a) timing b) energy usage model

Total consumed energy is:

$$E = \sum_{i=1}^{m} E_i + E_0 \tag{7}$$

The problem of time- and energy-efficient scheduling can be formulated as an integer linear programming model for minimizing schedule length:

$$\min T \tag{8}$$

or for the minimization of energy consumed:

$$\min E, \text{ subject to } T \leq T', \tag{9}$$

where T' is some limit on makespan. In both cases, it is further required that:

$$\sum_{i=1}^{m} \alpha_i = V \tag{10}$$

$$\sum_{i=1}^{j} x_i t^S + C \sum_{i=i}^{j} \alpha_i + t_j^R \leq T \qquad \forall j = 1, \ldots, m \tag{11}$$

$$t_i^I + t^S x_i + C\alpha_i + t_i^R = T \qquad \forall i = 1, \ldots, m \tag{12}$$

$$\alpha_i \leq V x_i \qquad \forall i = 1, \ldots, m \tag{13}$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, \ldots, m \tag{14}$$

In the above formulations, the sum of all load chunk sizes must be equal to the whole load (10). Inequality (11) ensures the proper timing: the startups and communications of all machines $M_1, \ldots, M_{j-1}$ have to occur before machine M_j starts up. The startups, communication and computation of machine M_j must end before the end of schedule T. Note that constraint (11) is an implicit equivalent of the optimality criterion used in DLT. Equation (12) allows to calculate the value of idle time t_i^I. Inequality (13) sets x_i to 1 if machine M_i is used in the schedule. With such a representation we can calculate the energy consumption in the same model. When minimizing schedule length with the objective function (8) and constraints (10)–(14) energy E necessary for that schedule is calculated from (7). When minimizing the energy usage with the objective function (9), and constraints (10)–(14) schedule length T have to be given, as we cannot perform direct bicriterial optimization with the above formulation.

5 Time-Energy Trade-off in Close-up

In our model we have a set of 12 parameters: $V, C, t^S, P^S, P^I, P^N, k^0, k, a^0, a$, size of the RAM indirectly represented by l and b, and machine number m. Testing all relations between all possible values of these parameters is not doable in the limited space of this paper. Thus, we decided to stick to the analysis of the relationships between schedule length and energy consumption. In the following charts all parameters have fixed values except for m and one parameter the

impact of which will be analyzed. With that setting we generate a series of (T, E) values using increasing number of available machines. A guide to the analyzed parameter ranges is given in Table 3.

Now let us discuss shortly the default values we used in our analysis. The units used are seconds, MegaBytes, Watts. The size of the load $V = 10000$ is 10GB. The value of $C = 0.006$ means that 1MB of data will be transfered in 0.006s and represents network with bandwidth ca. 1300Mbit/s. Power consumption values $P^I = 6, P^S = 101, P^N = 91$ are chosen from the range of values observed on real machines in the experiments described in Section 3. The power rate at the startup $P^I = 6$ and the startup time $t^S = 70$s represent computers waking up from hibernation to HDD, and then loading system or other necessary software. This is again a real measured value and the time of 70s is acceptable in schedules of lengths usually between 1000s and 10000s. Unless stated otherwise the above values were used for all charts. Parameters describing processing rate and energy cost of computations $a^0 = 0.08, a = 2.37, k^0 = 13.00, k = 294.43$ were chosen from the range of measured values presented in Table 1. The machine simulated in this analysis had 996MB of RAM available for the data. This may represent a 1GB machine with lightweight operating system and software, or for example 1.5GB VM or even 2GB machine with much heavier environment. Values of $l = -280303.72$ and $b = -2274.89$ where calculated to represent RAM of 996MB.

To obtain data for the charts presented in this study, the ILP model proposed in the previous section was programmed in CPLEX 12.6 software. Time minimization (8) was performed with changing the number of machines available for the computations: every point in a chart is the result of solving one minimization problem. Usually values of m up to 20 or 30 were tested, because larger values increase solution time beyond a few minutes per optimization. Since changes of m are of discrete nature, the corresponding points in the charts are connected with dashed lines to show the data series. We start with a close analysis of the relationship between makespan and energy cost for a given number of machines. Thus, for the selected numbers of machines we examined minimum energy derived from (9) with increasing schedule length. As this change is continuous in nature (any arbitrary time value can be used) we marked these schedules with solid lines.

In Fig. 3 energy and makespan for two different problem sizes V are depicted. This figure will suit us to discuss some phenomena, but also to explain how to read the following charts, as they contain many different curve shapes. The diagonal dashed lines for $V = 5000$ and $V = 10000$ mean that allowing more machines for computations was both decreasing schedule length and energy used. However, the curves have pipe-like shape, i.e. the set of points on the left end of the curve forms a vertical line. This means that allowing more computers than in the point at the bottom of the pipe did not shorten schedule. The reason is simple: it is impossible to use more machines. Machines start in sequence; machine M_j can start computations at time $\sum_{i=1}^{j} x_i t^S + C \sum_{i=i}^{j} \alpha_i$ and at some number of machines j this time exceeds T. Thus, the machines that were available, but not used, were only wasting power P^I in the entire time T. Note that the maximal number of usable computers changes with V and should be set prior to computations to save the energy. Due to the sequential starting of the machines the loads α_i are uneven. Consequently, for both data series (the dashed lines of the shortest schedules) virtually all machines performed out-of-core computations. For example, for $V = 10000$ only the last two machines are computing on-core. If we increase schedule length, more machines can get a load equal to 996MB of RAM and compute on-core which is energetically cheaper. For $V = 10000$ it is possible to give 996MB to at most nine machines and 1036MB to the remaining one, in this way obtaining the schedule with the lowest energy marked on the chart by a triangle. Between the point of the shortest time (bottom of the pipe) and the point of the lowest energy (triangle) we have a line of the minimal energy for a given schedule length. This line can be understood as a trade-off line where we can reduce computation time at the cost of increased energy consumption. And vice versa, we can reduce energy intake, but at the cost of longer processing.

Now let us zoom on the area of points representing the shortest schedules with 7 to 10 machines and follow the analysis in Fig. 4. We extended the line representing the minimum energy schedules from the point of the lowest energy to the right. It is visible that the energy grows with time; this is due to the idle time of machines in the schedule. Actually

Table 3 Index of the analyzed parameter ranges

Parameter	Unit	Typical value	Range		Studied in Fig.
			Min	Max	
V	[MB]	10000	200	100000	3, 7
C	[s/MB]	0.006	0.00001	0.1	8
t^S	[s]	70	0.1	100	9
P^S	[W]	101	101	112	-
P^I	[W]	6	6	79	-
P^N	[W]	91	91	116	-
k^0	[J/MB]	13.00	9.03	18.72	10
k	[J/MB]	294.43	150	500	11
a^0	[s/MB]	0.08	0.025	0.4	10
a	[s/MB]	2.37	0.53	2.37	-
ρ	[MB]	996	100	100000	12

with every second $(m+1) * P^I$ of energy is added. This also separates the region of infeasible solutions below the line. Similar lines have been observed for smaller machine numbers. For $m = 9$, the energy savings that can be achieved by the time-energy trade-off are more limited. Due to the sequential start of the machines, it is not possible to build a better schedule than with only two machines operating on-core. Similarly, for $m = 8$ and $m = 7$ no energy savings below the level achieved at the shortest schedule are possible by lengthening the schedules. Here the schedules with some machines operating on-core use more energy because of overloading the machines computing out-of-core.

Figure 5 shows these phenomena for schedules on 9 to 15 machines with even higher resolution. It is visible now that the time-energy trade-off line for $m = 10$ has two knees. Let us trace the line rightward from the point of the shortest makespan. We start in the steepest descent area, where only the last two machines are

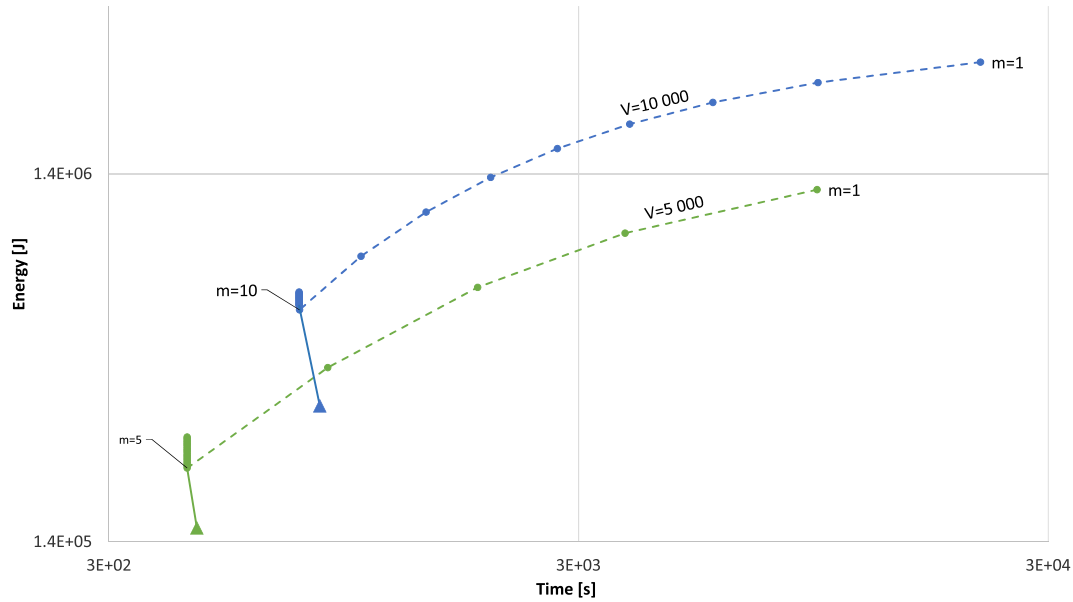

Fig. 3 Minimum time and minimum energy points for different load sizes (log scale)

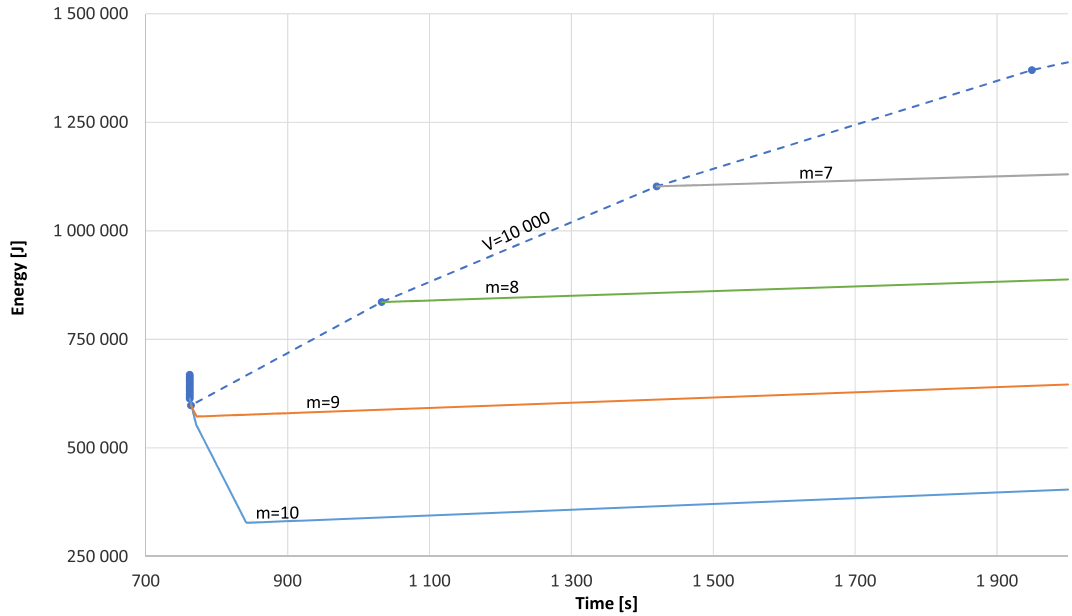

Fig. 4 Minimum energy line close-up for 7 to 10 machines

receiving loads α_i smaller than their RAM size, again due to sequential start of machines. Lengthening the schedule allows to shift the load processed out-of-core in the eight first machines to the last two machines. When the schedule is long enough, the penultimate machine receives a load of the RAM size and only the last machine can receive a load smaller than the RAM size, then the minimum energy line reaches its first knee. The process of shifting the load to the last machine, still working on-core, continues with increasing schedule length until arriving at the second knee (being the point of the lowest energy). Here the last machine receives a load equal to its RAM size. The line is steeper in the area where the RAM

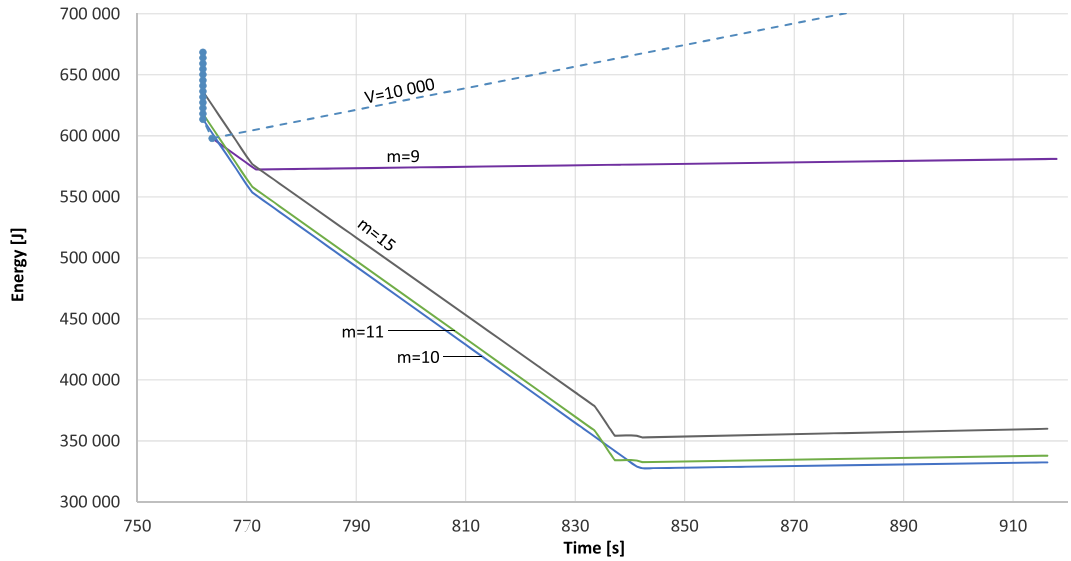

Fig. 5 Time-energy trade-off close-up for 9 to 15 machines

of the two last machines is being progressively filled with load than in the area where this happens only to the last machine. Another interesting effect can be observed with the minimum energy line for $m = 11$. The machines can finally conduct all the computations on-core, but the schedule must be long enough to start $m = 11$ processors. When such length is reached, the line gets another bend, and in a small area computations on 11 machines are more energy-efficient than on 10. All the lines for $m > 11$, here portrayed only for $m = 15$, expose the same shape because it is not effective to include more than 11 machines in the schedule. The lines differ only by an offset of the energy used by the excessive idle machines.

Figure 6 presents a system where machines start quicker, i.e. in $t^S = 7$. The rest of the parameters remain unchanged. The figure shows minimum energy lines for $m = 10$ to $m = 15$ machines. The (dashed) line of the shortest schedules turns up slightly milder than before which will be discussed in the next section (see Fig. 9). As the startup is shorter it is possible to use more machines in the schedules and thus reduce the makespan. As previously, the (solid) lines of the time-energy trade-off start at the points of the shortest schedule. The shapes of the minimum energy lines for $m = 10$ and $m = 11$ are similar as in the previous examples; however for $m > 11$ new phenomena emerge. Previously, straight lines run from the knee of the lowest energy to the rightmost part of the chart. Now they have segments where the minimum energy is stair-casing down. Note that the steps and points of minimum energy always happen at the same schedule length T. Each step represents excluding one machine from the schedule. The size of the drop in energy is $t^S(P^N + P^S) - 2t^S P^I$ which is the energy consumed by starting a machine and the originator waiting for the machine to start. The number of steps depends on the number of machines that can be switched off to reach the lowest energy schedule. Here, the lowest energy is achieved at $m = 11$, with all machines operating on-core. Thus, e.g. the schedule with $m = 12$ can switch off one machine, giving it one drop, and so on.

6 Impact of Other Parameters on the Time-Energy Trade-off

In this section we will trace the relation between time and energy consumption in a broader scene. Therefore, we will analyze mainly the shortest schedules, skipping in some charts the whiskers of the time-energy trade-off for clarity. The time-energy trade-off will be still present in the discussion; however we will

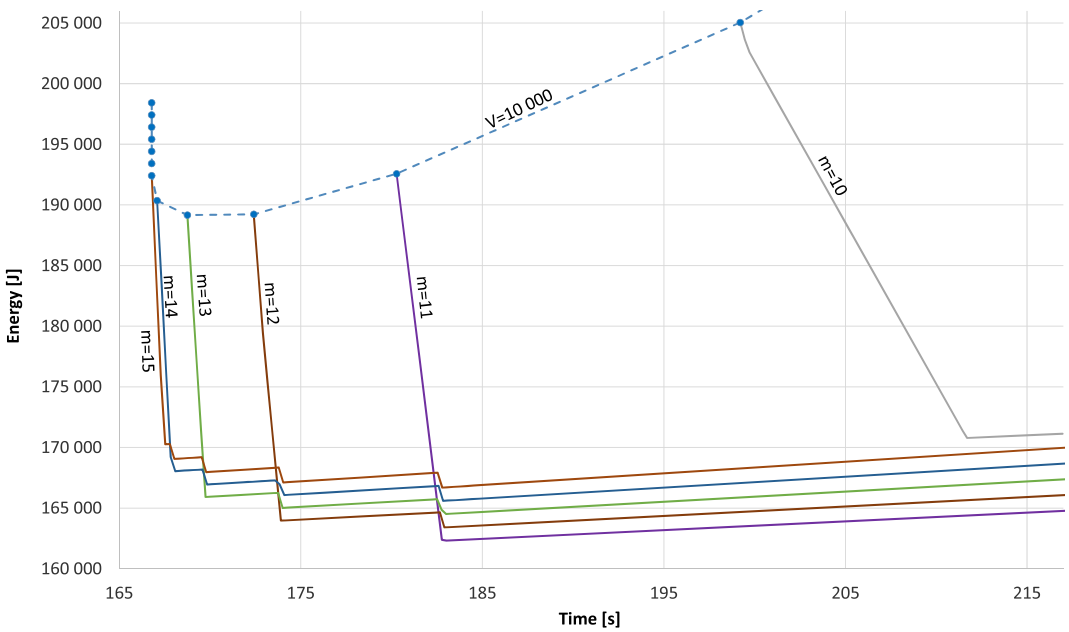

Fig. 6 Time-energy trade-off for systems with shorter startup $t^S = 7$

focus on the impact of the system parameter changes on the performance of the computations.

In Fig. 7 we analyze the impact of the problem size V. For $V = 200$ we observe a straight vertical line, perpendicular to the time axis. This means that the time of the computations could not be improved by applying more machines while the energy cost was still growing. The reason is very simple: there was not enough load to exploit more than one computer and all machines beyond M_1 were merely wasting energy in the idle state for the entire time T. The lines for $V = 5000$ and $V = 10000$ were discussed previously. For $V = 50000$ and especially for $V = 100000$ the curve becomes nearly horizontal, i.e. almost perpendicular to energy axis. In this area it is possible to significantly shorten computations by adding more machines, but savings in the energy will be very limited because for $V > 1000$ virtually all machines perform out-of-core computations.

In Fig. 8 we can observe how different values of communication rate C affect the computations. For the slowest communication at $C = 0.1$, the number of machines that can receive a load is the smallest ($m = 8$) and the schedules are longer and more energy-consuming. We get better results with $C = 0.01$ and C=1E-5; however the difference between them seems small considering the range of change in the network speed. The curves for communication rate C <1E-5 are not visible in the chart because they overlap the curve of C =1E-5. This shows that there are limitations of speeding the computations up and reducing the energy usage by means of improving only the networking capabilities.

Startup time t^S is a parameter limiting the number of machines that can be included in the schedule. In the previous charts the significant value of t^S was one of the main bottlenecks preventing the increase of machine number. In Fig. 9 we study the effect of changing t^S on the performance of the computations. For $t^S = 100$ it is possible to use only 8 machines and we can see a very clear pipe-like shape in Fig. 9. For $t^S \geq 50$ the shortest schedule is also the one with the minimum energy. However, for $t^S \leq 10$ the minimum energy is reached before hitting the limit of the number of machines that can be used in the schedule. Although it could be expected that the minimum energy is achieved when all, or as many as possible, machines receive a load small enough to process it in RAM, it is not the case. For $t^S = 10$ half of the machines still perform out-of-core computations in the minimum energy schedule. In the case of $t^S = 1$ it grows to 28 machines, and the pipe shape is more outstretched, but the limit still exists. With decreasing t^S the share of machines computing out-of-core shrinks, but actually never disappears. For really small startups, such as $t^S = 0.5$ and $t^S = 0.1$, the impact of

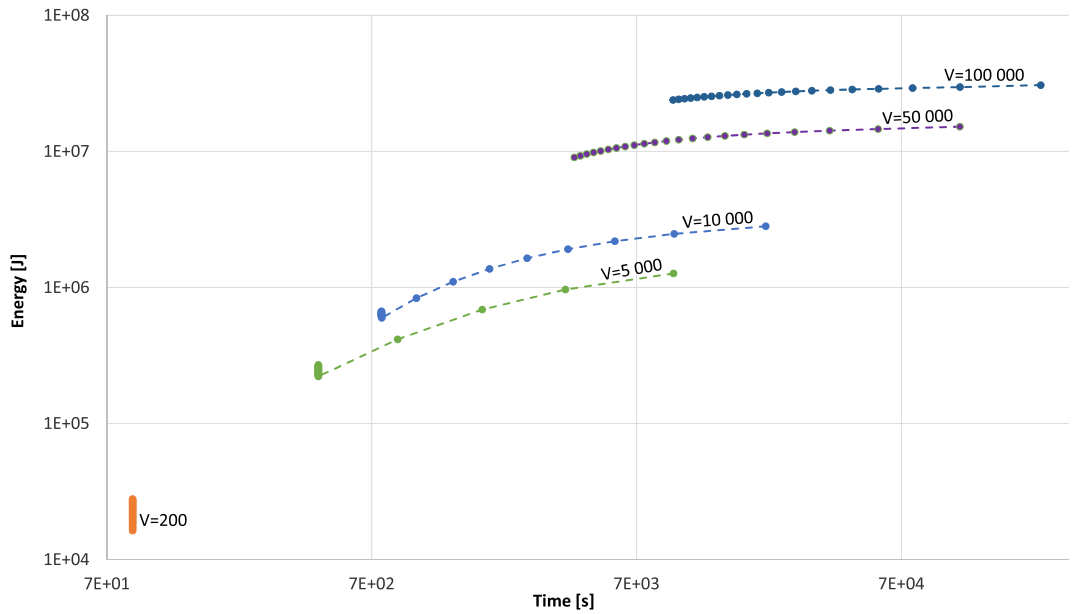

Fig. 7 Energy vs Time for different load sizes V

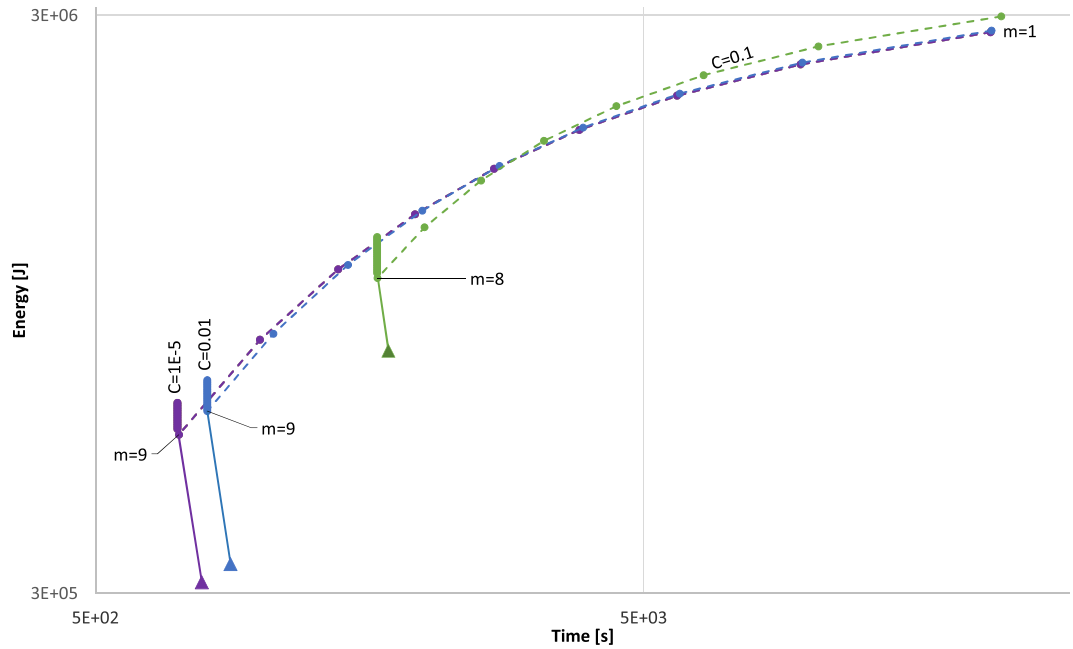

Fig. 8 Energy vs Time for different communication rates C

the startup time on the number of usable machines is milder as we see no sharp upright pipe shape. Since the curves noticeably bend upward at their leftmost ends it means that the shorter computation time is gained at the cost of increased energy consumption. Yet, this depends on the load size V (see Fig. 7).

The impact of computation rate a^0 is shown in Fig. 10. Let us observe that computation rate a^0 and energy cost k^0 are not independent. Decreasing a^0 means shorter computation and consequently smaller energy consumed per load unit. Yet, as observed in Section 3 this relationship is not linear because

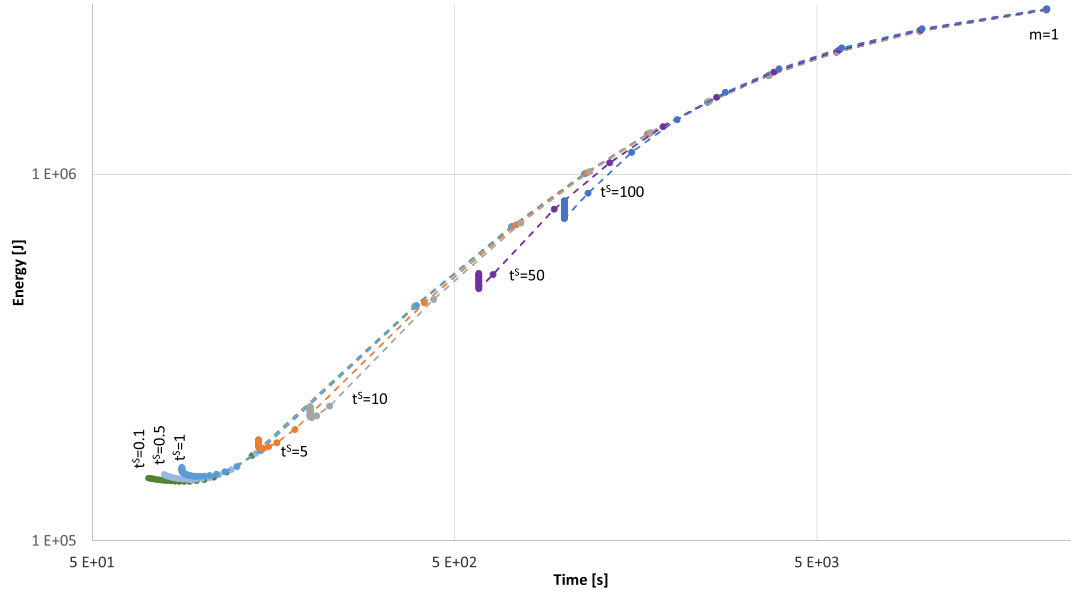

Fig. 9 Energy vs Time for different startup times t^S

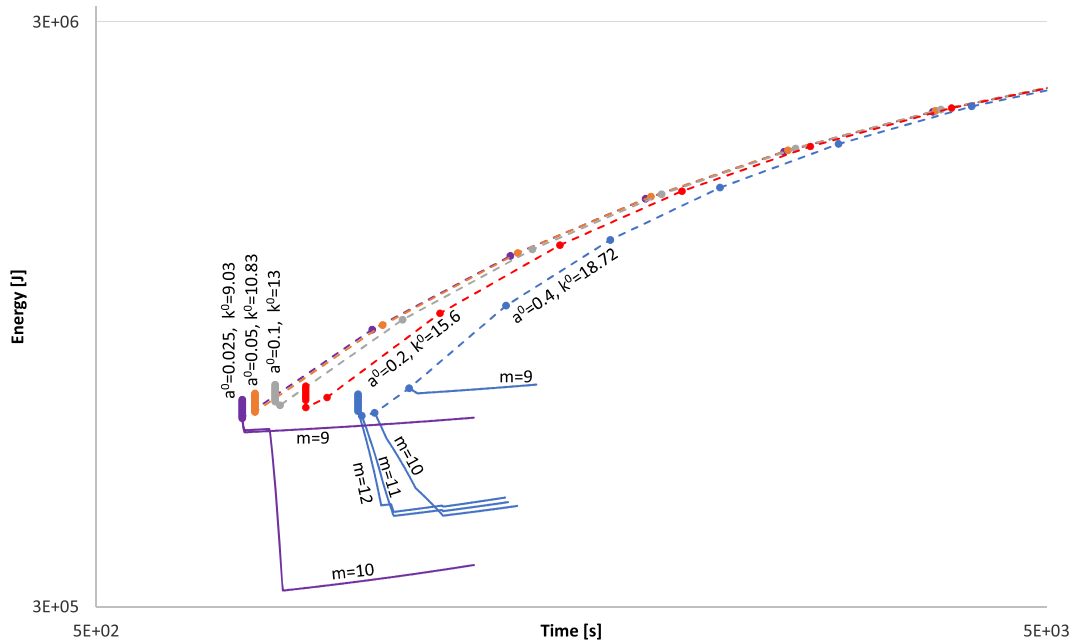

Fig. 10 Energy vs Time for different on-core computation rates a^0

computer systems are not energy-proportional. For example, power consumption of the computing equipment does not halve with dividing CPU speed by two. Therefore, with halving a^0 (i.e. doubling the speed) we divided k^0 by 1.2, starting with $a^0 = 0.1, k^0 = 13$ as reference values. Figure 10 zooms in on the area of pipes, where the most interesting observations can be made. The processing rate here affects the number of machines that can be used in a schedule. For the fastest processing at $a^0 = 0.025$ and $a^0 = 0.05$ it is $m = 9$, and it increases for the cases of slower processing, up to $m = 12$ at $a^0 = 0.1$. For slower processing there are visible schedules with lower energy which may seem to be a paradox. The explanation is as follows. For $m = 9$ (marked on chart) still all processing is done out-of-core, and with slower computations more machines can be included into the schedule and the load is divided into smaller chunks. Thus, on some machines it is fitting into memory, in effect allowing more energy-efficient on-core computations. Now let us discuss whiskers marking the lines of minimum energy. For clarity of the chart, whiskers are shown only for the border values $a^0 = 0.4$ and $a^0 = 0.025$, and are drawn only partially in the area of the linear growth. Careful readers will easily notice that the shapes of the whiskers for $a^0 = 0.4$ resemble the ones discussed earlier. The minimum energy curve for $m = 9$ has the same shape as in Fig. 5 and for the cases of $m = 11, m = 12$ there are staircase patterns as in Fig. 6. The curve for $m = 10$ has one more bending point. The curve is changing its slope three times, because in the shortest schedule there were three machines with load α_i smaller than RAM size. This allowed to off-load the out-of-core computations and fill the three machines up to the RAM size in three stages of increasing the makespan. However, a completely new shape of minimum energy line can be observed for $a^0 = 0.025$ and $m = 10$. Within the minimum length schedule it is not possible to include the tenth machine into computations. Still, for its minimum energy line two steps can be seen. Firstly it obtains energy minimum using only 9 machines. The line is slightly higher than the one for $m = 9$ because of the idle energy of the tenth machine. Then after adding some more time to the makespan the tenth machine can be included in the computation, reaching the energy-optimal energy point for the computation rate $a^0 = 0.025$.

In Fig. 11 the impact of changing the energy cost of out-of-core computations k on total energy consumption is depicted. Out-of-core computations are the costliest part of the analyzed schedules. The startup time was shortened to $t^S = 7$ to allow better insight into the discussed phenomena. At a given value of

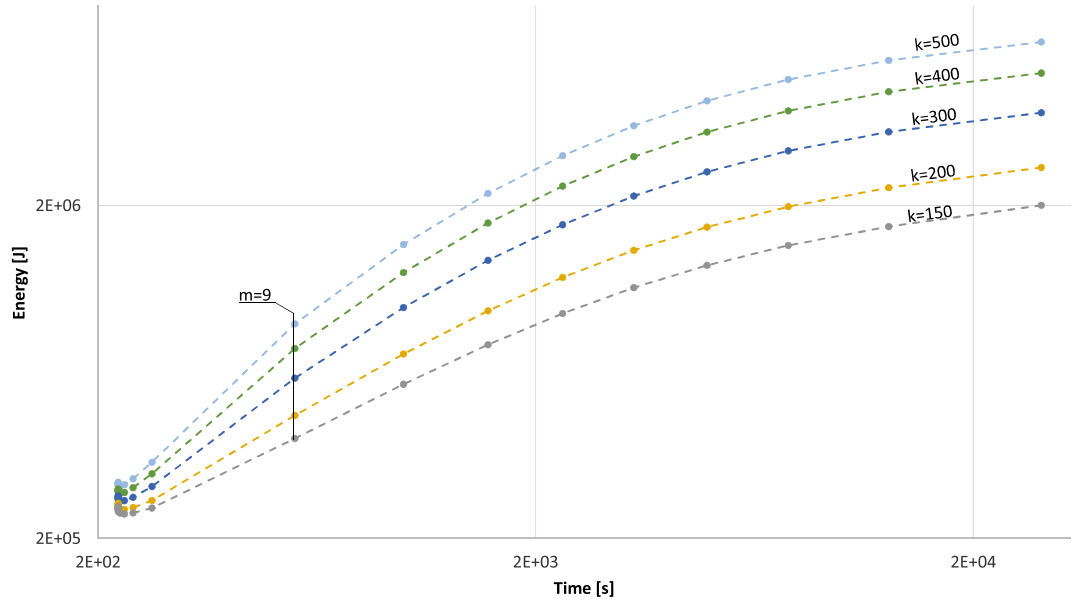

Fig. 11 Energy vs Time for different energy factors k in out-of-core computations

m the points on all the curves align to vertical lines. There are the same minimum schedule lengths for the same number of machines, as k has no impact on the schedule length. If we treat the $k = 300$ curve as a reference, then we have systems with costlier out-of-core computations $k = 400$ and $k = 500$, as well as less costly systems at $k = 200$ and $k = 150$. Yet, all of the curves converge to a much smaller difference in energy consumption with an increasing number of machines. Part of this happens before loads α_i start to fit into RAM; even on $m = 9$ all machines still work out-of-core. The savings in energy usage are a sheer result of the parallelism shortening the schedule. For bigger values of m the convergence

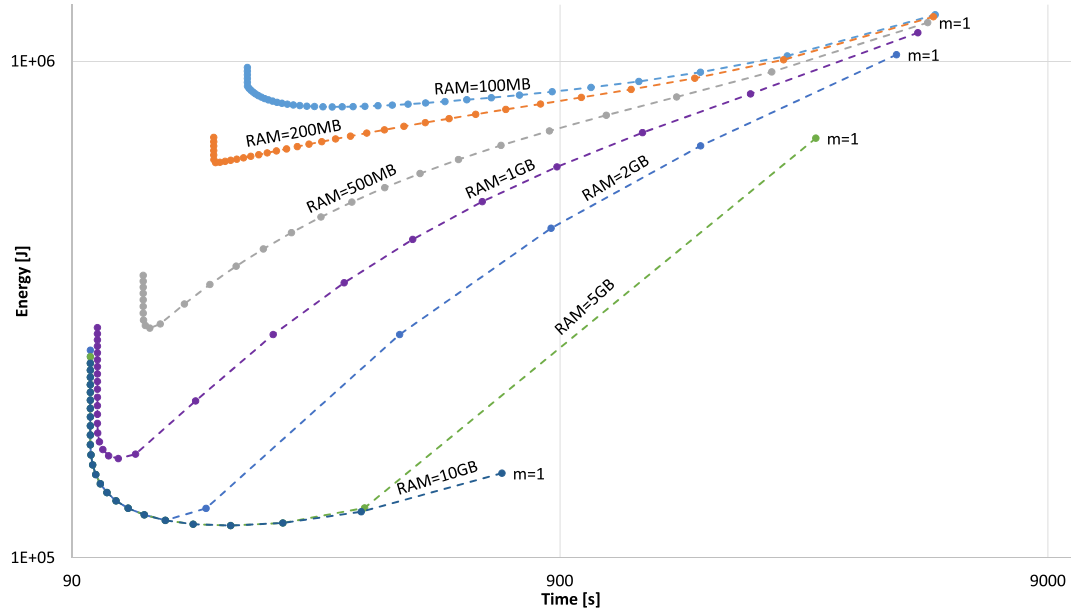

Fig. 12 Energy vs Time for different RAM sizes ρ

is even more apparent because more machines operate on-core. This shows that the reductions of the energy intake of the equipment have limits. Oppositely, energy costs of worse hardware can be often made up with better parallelism.

Figure 12 represents an example of a slightly different configuration. Parameters $a^0 = 0.066, a = 0.53$ represent a faster machine with SSD drive. This results in higher energy demand $k^0 = 9.03, k = 82.66$, $P^S = 112$, $P^N = 116$. Machines are waiting almost ready to start processing with $t^S = 5$ and $P^I = 79$. Also communication is faster, $C = 0.002$, which represents effective connection speed of ca. 4000 Mbit/s. The parameter changed in Fig. 12 is RAM size ρ on the machines. Parameters b and l were changing accordingly. For ρ =10GB, all the data fit into RAM. The curve here has a quite wide area of time-energy trade-off. For ρ =5GB and ρ =2GB with increasing number of computers the chunks of load start fitting into memory, and the curves are overlaying with the one for ρ =10G. For the remaining four sizes of RAM, we observe pipes similar to the previous charts. Still, there are differences: the number of machines that can be included into the schedule before the curve turns upright greatly increases. This also broadens the area near the optimum energy, from a few machines up to 20 for RAM=100MB.

7 Conclusions

In this paper the problem of scheduling divisible loads for the criteria of energy and makespan in homogeneous systems with memory hierarchy has been considered. Although all of the computer systems have hierarchical memory, the memory hierarchy is still novel in the performance and scheduling models of parallel computations. Models always simplify reality at some point, but at the same time allow to gain new insights into the studied systems. The scheduling model proposed in this paper turns out a valuable instrument in analytically modeling the performance of distributed systems. The performance evaluation has shown that there is a trade-off between energy cost and schedule length. The trade-off as well as the overall performance is ruled by a complex interplay between the speed and power of computing on-core vs out-of-core, costs of activating new machines, communication delays, and the size of the solved problem. Overall, it can be concluded that in the wide ranges of system parameters parallel processing has a synergistic effect on energy and makespan: it is possible to economize on both criteria by adding new machines. However, this phenomenon is limited to big size computations and short startup times. Bigger startup times quickly cut off chances for time and energy savings. Thus, we could say that parameters defining the parallelism will strongly affect optimality of the schedules length, as well as energy used, and should be chosen carefully before the computations. Moreover, it could be observed that the energy savings obtained by the change of one parameter, or one part of equipment, are usually limited. Progress in all areas is needed for a steady reduction of power consumption required to fuel high performance computations and big datacenters.

In this paper we examined homogeneous systems. Therefore, heterogeneous systems are an attractive subject of the future work. An interesting case is created by datacenters with a few types of machine instances. Designing time and energy-efficient scenarios of partitioning datacenters into several hibernated, sleeping, idle waiting, and running machines is another challenging research area. A very rudimentary load distribution method has been assumed. Hence, the impact of more advanced load distribution algorithms and networks should be considered.

Acknowledgments Jakub Marszałkowski acknowledges support by the FNR (Luxembourg) and NCBiR (Poland), through IShOP project, INTER/POLLUX/ 13/6466384.

References

1. Abdullah, M., Othman, M.: Cost-based multi-QoS job scheduling using divisible load theory in cloud computing. Procedia Comput. Sci. **18**, 928–935 (2013)
2. Abello, J., Scott Vitter, J. (eds.): External Memory Algorithms. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. AMS (1999)

3. Agrawal, R., Jagadish, H.: Partitioning techniques for large-grained parallelism. IEEE Trans. Comput. **37**(12), 1627–1634 (1988)
4. Apache Software Foundation: Hbase and MapReduce. https://hbase.apache.org/book/mapreduce.html (2014)
5. Berlińska, J., Drozdowski, M.: Scheduling divisible MapReduce computations. J. Parallel Distrib. Comput. **71**(3), 450–459 (2011)
6. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos (1996)
7. Carter, J., Rajamani, K.: Designing energy-efficient servers and data centers. Computer **43**(7), 76–78 (2010)
8. Cheng, Y., Robertazzi, T.: Distributed computation with communication delay. IEEE Trans. Aerosp. Electron. Syst. **24**(6), 700–712 (1988)
9. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: OSDI'04: Sixth Symposium on Operating System Design and Implementation, pp. 137–150 (2004)
10. Dorronsoro, B., Danoy, G., Bouvry, P.: Special issue: Energy-efficiency in large distributed computing architectures. Futur. Gener. Comput. Syst. **36**, 187–188 (2014)
11. Doulkeridis, C., Norvag, K.: A survey of large-scale analytical query processing in MapReduce. VLDB J. **23**(3), 355–380 (2014)
12. Drozdowski, M.: Scheduling for Parallel Processing. Springer-Verlag New York Inc (2009)
13. Drozdowski, M., Marszałkowski, J.M., Marszałkowski, J.: Energy trade-offs analysis using equal-energy maps. Futur. Gener. Comput. Syst. **36**, 311–321 (2014)
14. Drozdowski, M., Wolniewicz, P.: Experiments with scheduling divisible tasks in clusters of workstations. In: Bode, A., Ludwig, T., Karl, W., Wismuller, R. (eds.) Proceedings of the 6th International Euro-Par Conference on Parallel Processing, LNCS 1900, pp. 311–319. Springer-Verlag (2000)
15. Drozdowski, M., Wolniewicz, P.: Out-of-core divisible load processing. IEEE Trans. Parallel Distrib. Comput. **14**(10), 1048–1056 (2003)
16. Fuller, S., Millett, L.: Computing performance: Game over or next level? Computer **44**(1), 31–38 (2011)
17. Ghanbari, S., Othman, M.: Comprehensive review on divisible load theory: Concepts, strategies, and approaches. Math. Probl. Eng. **2014** (2014)
18. Hunter, H., Lastras-Montano, L., Bhattacharjee, B.: Adapting server systems for new memory technologies. Computer **47**(9), 78–84 (2014)
19. Iyer, G.N., Veeravalli, B., Krishnamoorthy, S.G.: On handling large-scale polynomial multiplications in compute cloud environments using divisible load paradigm. IEEE Trans. Aerosp. Electron. Syst. **48**(1), 820–831 (2012)
20. Jose, J., Potluri, S., Subramoni, H., Lu, X., Hamidouche, K., Schulz, K., Sundar, H., Panda, D.: Designing scalable out-of-core sorting with hybrid MPI+PGAS programming models. In: Proceedings of 8th International Conference on Partitioned Global Address Space Programming Models, p. 9. ACM (2014)
21. Katz, R.: Tech titans building boom. IEEE Spectr. **46**(2), 40–54 (2009)
22. Kogge, P.: The tops in the flops. IEEE Spectr. **48**(2), 48–54 (2011)
23. Kundeti, V., Rajasekaran, S.: Efficient out-of-core sorting algorithms for the parallel disks model. J. Parallel Distrib. Comput. **71**(11), 1427–1433 (2011)
24. Lee, K.H., Lee, Y.J., Choi, H., Chung, Y.D., Moon, B.: Parallel data processing with MapReduce: A survey. ACM SIGMOD Rec. **40**(4), 11–20 (2011)
25. Li, X., Veeravalli, B., Ko, C.: Distributed image processing on a network of workstations. Int. J. Comput. Appl. **25**(2), 10 (2003)
26. Lin, J., Dyer, C.: Data-Intensive Text Processing with MapReduce. Morgan & Claypool (2010)
27. Mei, X., Zhao, K., Liu, C., Chu, X.: Benchmarking the memory hierarchy of modern GPUs. In: Network and Parallel Computing, pp. 144–156. Springer (2014)
28. Mills, R.T., Yue, C., Stathopoulos, A., Nikolopoulos, D.S.: Runtime and programming support for memory adaptation in scientific applications via local disk and remote memory. J. Grid Comput. **5**(2), 213–234 (2007)
29. Moges, M., Ramirez, L., Gamboa, C., Robertazzi, T.: Monetary cost and energy use optimization in divisible load processing. In: Proceedings of the 2004 Conference on Information Sciences and Systems, p. 6. Princeton University (2004)
30. MongoDB Inc: Mongo db manual 2.4. http://docs.mongodb.org/manual/core/map-reduce/ (2014)
31. Nesmachnow, S., Dorronsoro, B., Pecero, J.E., Bouvry, P.: Energy-aware scheduling on multicore heterogeneous grid computing systems. J. Grid Comput. **11**(4), 653–680 (2013)
32. Plattner, H.: Changes in hardware. In: A Course in In-Memory Data Management, pp. 23–32. Springer (2014)
33. Qureshi, M.B., Dehnavi, M., Min-Allah, N., Qureshi, M.S., Hussain, H., Rentifis, I., Tziritas, N., Loukopoulos, T., Khan, S., Xu, C.Z., Zomaya, A.: Survey on grid resource allocation mechanisms. J. Grid Comput. **12**(2), 399–441 (2014)
34. Robertazzi, T.: Ten reasons to use divisible load theory. Computer **36**(5), 63–68 (2003)
35. Robertazzi, T.: Divisible Load Scheduling. http://www.ece.sunysb.edu/tom/dlt.html (2014)
36. Scott Vitter, J.: External memory algorithms and data structures: Dealing with massive data. ACM Comput. Surv. **33**(2), 209–271 (2001)
37. Sohn, J., Robertazzi, T., Luryi, S.: Optimizing computing costs using divisible load analysis. IEEE Trans. Parallel Distrib. Syst. **9**(3), 225–234 (1998)
38. Sun, G.: Exploring Memory Hierarchy Design with Emerging Memory Technologies, Lecture Notes in Electrical Engineering, vol. 267. Springer International Publishing (2014)
39. Swanson, S., Caulfield, A.: Refactor, reduce, recycle: Restructuring the I/O stack for the future of storage. Computer **46**(8), 52–59 (2013)
40. White, T.: Hadoop: The Definitive Guide. O'Reilly (2012)
41. Ye, Y., Du Zhihuiand Bader, D., Yang, Q., Huo, W.: GPUMemsort: A high performance graphic co-processors sorting algorithm for large scale in-memory data. GSTF Int. J. Comput. **1**(2) (2011)
42. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. In: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, pp. 10–10 (2010)