Thermal aware scheduling on distributed computing water heaters

Issam Rais, Eddy Caron and Laurent Lefevre

Inria - LIP Laboratory, ENS, University of Lyon, France
{Issam.Rais, Eddy.Caron, Laurent.Lefevre}@inria.fr

***Abstract*—Combining computation and water warming is an efficient alternative for recycling dissipated energy. Based on a set of Computing Water Heaters, used as a distributed super computer, we introduce a computing water heater scheduling system that has been validated by simulation on extreme workloads. These simulations lead us to reach an approximate CPU usage of about 60% while minimizing the thermal resistance usage to 10% (only when necessary) on a complex steady state.**

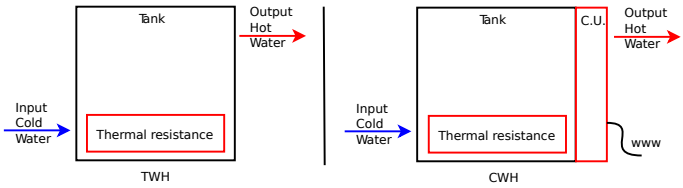

Fig. 1: from TWH to CWH

I. Introduction

Energy consumption is a real limiting factor for large scale data centers and supercomputers [3]. For the cooling of such systems, generated heat must be processed through air, water or free cooling facilities. At the same time, there exists a real need for hot water which represents a huge part of consumed energy by households. The idea here is to combine the need of water warming and the need to compute. The proposed architecture explores potential energy reduction by supporting a set of distributed computing water heaters (CWH), recovering dissipated energy from computation directly into hot water for household usage.

We collaborate with a CWH industrial provider (the deFab company) who needed a scheduling framework.[1] Computing needed by a customer has to be efficiently delivered and scheduled by the "deFab Platform" to one of the machine composing the distributed set of computing water heaters.

The main contributions of this paper consists in:

1) choosing a coherent scheduling architecture compatible with a set of distributed computing entities and with the needs of the multiple users of the system;
2) proposing adapted multi level scheduling policies and heuristics while properly fulfilling constraints imposed by a set of computing water heaters;
3) studying crippling and steady state cases that could happened in such infrastructures through simulation.

II. Computing Water Heaters (CWH)

A Computing Water Heater (CWH) differs in many ways from a Traditional Water Heater (TWH) (Figure 1).

A. Architecture

A TWH is an entity composed of a tank of water built together with a thermal resistance and respecting standards for inputting and outputting water. The first valve represents the arrival of water. Once a water pipe is connected, cold water can be inputted. The second one is for the output pipe. Once the water has been heated, through the usage of the thermal resistance, hot water can be outputted for common usage.

As shown in Figure 1, a CWH differs from a TWH in an architecture level. In fact, a CWH is built with a Computing Unit (noted C.U. on Figure 1) which warms the water. This CU is connected to the Internet to receive computational tasks.

B. Water life cycle

Globally, a TWH has only one goal: heat water to reach a target temperature set at a factory level. When hot water is requested and delivered, cold water is inputed, making the temperature of water in the tank drop and at the same time, leaving the tank at the same volume of water. This newly inputted cold water will be heated by the thermal resistance and outputted when someone will ask for hot water.

A CWH has approximately the same water life cycle as a TWH. It differs in the way of heating water. Instead of using only the thermal resistance, a CWH could use dissipated energy from the CU. This entity is composed of servers connected like a cluster, supervised by a machine delivering work. The CWH usage is specific:

- Availability of a CWH is based on the temperature of water in the tank. So the availability of a CWH is true if and only if temperature of water is below the target temperature.

[1]This work is supported by the ELCI FSN project ("Fonds pour la Société Numérique") project that associates academic and industrial partners to design and provide software environment for very high performance computing. We thank B. Laplane from the deFab company for his technical support.

- When a CWH reaches its target temperature, it stops computing, making the components of the CU unavailable to computation.
- Such a CWH is available again if and only if a water usage is witnessed and that the temperature of water decreases.

Several crippling cases can be encountered :

- Starvation : This case occurs when there is not enough work to reach the target temperature of every CWH by using only the CU
- Saturation : When a CWH reaches its target temperature, it stops requesting work. If no water usage is made, the temperature will remain the same, placing the CWH in a saturation state

III. A SCHEDULING ARCHITECTURE FOR CWH

A. Scheduling architecture description

A multiple hierarchical scheduling is proposed. Every scheduling point only knows the level underneath. The pull/push approach and the multiple hierarchical scheduling has been designed to lighten the decision making process of every decision making point in the architecture.

A customer based system implies an online scheduling. We consider here that all the incoming applications will be based on embarrassingly data parallelism with no constraints nor communications between tasks.

A scheduler manages a fixed number of machines. The scheduler asks work through a pull action. It is the one that delivers the work to the CUs. So a scheduler only communicates with the metaScheduler and the CUs under its management. The metaScheduler is the entity that answers the pull and stores work delivered by the customer. Due to the distributed aspect of the set of machines, the life cycle of these tasks has to be described.

B. The computing life cycle

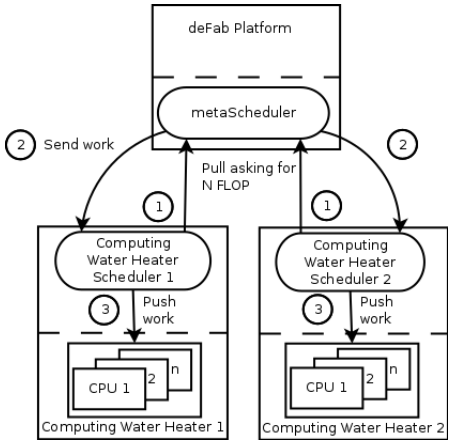

Fig. 2: The deFab hierarchical architecture

From the "deFab Platform" to CUs on CWH, Figure 2 presents the proposed architecture applied to the deFab infrastructure.

A user sends a request with all his needed computation. This computation is known and supported by the deFab infrastructure. The "deFab platform" determines the needed information for the metaScheduler concerning this application.

A Computing Water Heater Scheduler (CWHS) is the one to start communication. In function of temperature of water in the tank and in function of the computation power owned by the CWH, a "Pull" request will be delivered. This pull contains a number of FLOP for each core in the CWH. The number of FLOP asked represents the needed work to reach the target temperature of the CWH.

When the CWHS receives a task (arrow 2), and if there is some unused CU under its command, it sends it to the free unused CU as shown on the arrow 3. If there is no free CU, the received task is put on a queue (position of the arriving task in the queue depends on the chosen policy). Thus, the CWHS is also the one to deliver work to the CUs existing on the CWH.

The metaScheduler receives work from customers. Users upload needed files for their computation and applications are already deployed on every machine. It also is the one answering the pull requested by one of the CWHS from the deFab infrastructure. When the metaScheduler receives a pull, it tries to answer it as soon as possible. If there is enough work to answer to this request, metaScheduler starts to send chosen tasks.

The application model is based on the Expected Time To Compute Model [1]. We consider that the FLOP estimation is perfect (because the deFab infrastructure allows only known application to run, so we consider that the applications are very well known).

The supported scheduling is non preemptive and a task cannot leave the concerned CWH until it has finished its execution. Solutions like migrating extra tasks could be applied, while implying other problematics like good prediction of water usage.

IV. METASCHEDULER POLICIES

Two heuristics were developed for the metaScheduler :

1) First Requested First Served policy (FRFS): The goal of this heuristic is to maximize correspondence between the pull and the answer to this pull. Thus, maximizing the usage of CU on the CWH to reach the target temperature. This heuristic is also aiming for a maximization of priority so financial gain. When a pull is received, an immediate response is made, considering only available work. This heuristic is not fair when there is starvation. In fact, if there is enough work only for one CWH, all work will be delivered to the first puller while other won't receive any work.

2) Resource and Load Aware Policy (RLA): This heuristic allows the metaScheduler to be fair between all CWH. It also maximizes the usage rate of every core between every CWH. At the same time, it permits minimizing the inactivity of a CWH compared to another one. It relies the hypothetical temperature(the temperature that will reach the water in the tank if all the received task are executed if no water usage

is seen) of every CWH to fairly give work to the remaining CWH.

V. Results and validation

We study several cases that could append during a life cycle of a set of CWH (steady state, saturation and starvation). The SimGrid framework [2] was chosen as the base for the simulator.

The green curves of all the following graphs represent the accumulated power used by all components present on the CWH (CPUS and the thermal resistance). The black curves represents the progression of the temperature of the water in the CWH.

In the figures, the red part represents the usage, in percentage, of the CPUs during the simulation for a specific CWH. While the blue part represents time, in percentage, when the resistance is on. Finally, the green part represents the part of the time when the CPUs are waiting for work.

A. Starvation and saturation: complex scenarios

The following section presents how the implemented heuristics behave concerning complex scenarios (Section II).

1) First Requested First Served (FRFS) policy under starvation: The results shown in this section are made with a set of 10 CWH. Each one has 10 CPUs. The distributed platform has to perform a set of computing works.

Two metrics (power and temperature) from selected CWH are exposed. CWH 0 (Figure 3) is not getting any work from the metaScheduler and uses the thermal resistance to warm water, while CWH 2 (Figure 3) is getting work from metaScheduler and warms water through the CPU usage.

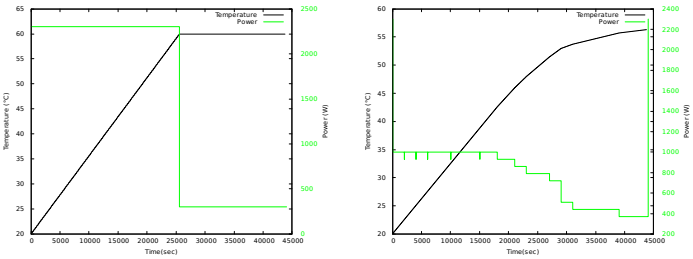

Fig. 3: Warming water of CWH 0 (through thermal resistance) and CWH 2 (through CPU usage)

FRFS properly pulls from CWH 1 and 2. Considering that these CWH are asking for enough FLOP, the metaScheduler is delivering all the pending work. All other CWH will be in a starvation state, and must heat their water using the thermal resistance, like CWH 0 (Figure 3). Figure 4 shows that only 2 out of 10 CWH received work.

2) Resource and Load Aware Policy (RLA) under starvation: This validation considers the same distributed infrastructure of CWH (10 CWH with 10 CPUs).

Figure 5 represents the CWH 2 behavior with RLA policiy, which benefits from CPU usage to warm water. But after second 6000, the thermal resistance is activated due to the lack of work left on the metaScheduler.

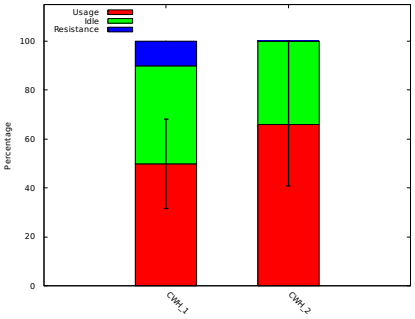

Fig. 4: CWH behavior during starvation (FRFS policy)

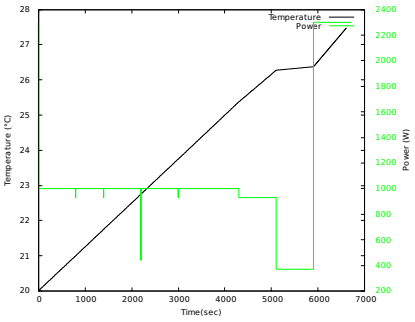

Fig. 5: CWH 2: warming water mostly through CPU usage

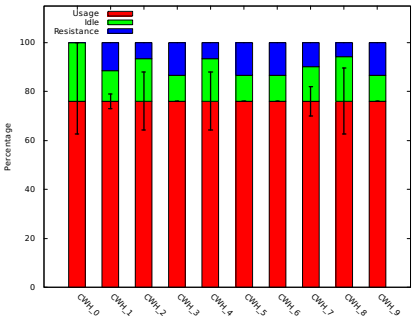

Fig. 6: CWH behavior during starvation (RLA Policy)

Figure 6 shows a balanced and fair distribution of work between Computing Water Heaters. Actually, the usage of the CPU is around 75% for every CWH.

It should be noted that under the same work demand and with the same set of CWH, FRFS finishes the simulation at $t = 45000$ while Resource and Load Aware Policy finishes the simulation at $t = 7000$. In other words, a reduction of about 85% of the completion time is observed with RLA under starvation.

3) Saturation: This section presents a simulation of 3 CWH, with a work demand superior to what it is needed to reach the target temperature of every CWH. Resource and Load Aware Policy is the chosen heuristic.

Figure 7 shows the evolution of temperature and power used during the simulation of CWH 0. A peak on the temperature curve shows a water usage on the CWH. The saturation state is reached around $t = 70000$. Some water usages occur

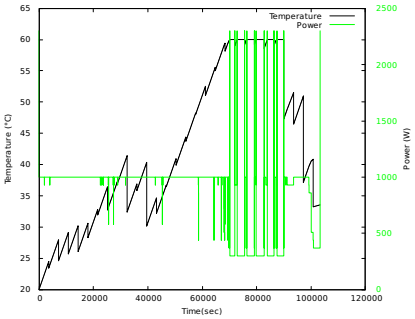

Fig. 7: CWH 0: with water usage

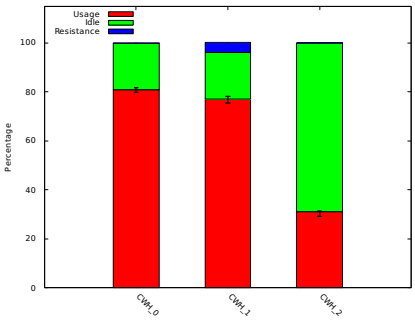

Fig. 8: CWH behavior during saturation (RLA policy)

during saturation, allowing extra work to be executed and thus, reaching again the target temperature.

This behavior can be observed on Figure 7, by analyzing the oscillations of power when the target temperature is reached. When all CUs received work, power used is then around $1000W$ (all CUs at work). A peak is then observed. It is the thermal resistance that is triggered to reach the target temperature.

Even with a CWH receiving enough work to reach the target temperature only with CPU usage, we can observe unequal CPU usage (Figure 8). It shows that the results concerning the usage of CPUs depends not only on the type of workLoad or the chosen heuristic, but also on the water usage.

B. Steady state: experimental validation scenario

The studied steady state presents a work demand representing an online work arrival for a set of CWH over 2 weeks.

Every CWH follows a realistic set of water usage which represents an usual water usage of a French household over a period of two weeks.

Figure 9 represents the evolution of temperature and power consumed by the representative CWH 9.

This figure shows that the heating of the set of CWH is achieved by thermal dissipation from CPUs. Besides the first two reach of the target temperature, the instantaneous consumption is around $1000W$ ($10 \times 100W$, 100W is the consumption of a CPU at full power). When the CWH almost reaches target temperature, peaks are observed. It corresponds to a water usage not important enough to deploy a task on a CPU without exceeding the target temperature.

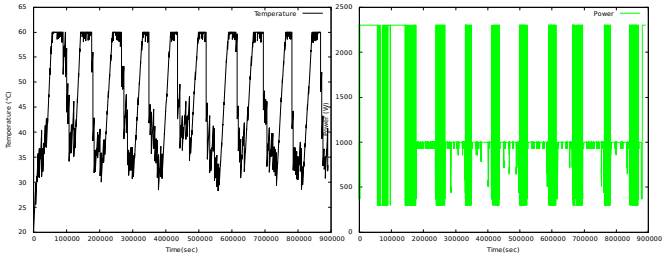

Fig. 9: Life cycle of a CWH : temperature and energy consumption

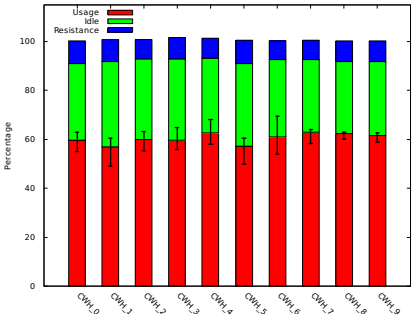

Fig. 10: CWH usage: steady state

Figure 10 shows a fair distribution of work. One can also note that a high rate of idle exists. It can be explained by the fact that a CWH doesn't ask work when its target temperature is reached. The part of resistance usage can be interpreted by the fact that the heuristics implemented on the CWHS chooses not to deliver a task when the execution of this task will make the water reach a temperature higher than the target temperature.

VI. Conclusion and Perspectives

In this work, we established an efficient and adapted scheduling system for a distributed infrastructure of computing water heaters. We implemented the scheduling architecture and relevant heuristics on a SimGrid based simulator.

The results obtained with various extreme and steady workloads, show that the chosen architecture and heuristic are compatible and efficient with the requirements of a set of computing water heaters. Thus, on a steady state, we reached an approximate CPU usage of about 60% and minimized the thermal resistance usage (10%, only when no computation is available on metaScheduler).

References

[1] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen. Task execution time modeling for heterogeneous computing systems. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pages 185–199. IEEE, 2000.

[2] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter. Simgrid: a sustained effort for the versatile simulation of large scale distributed systems. *CoRR*, abs/1309.1630, 2013.

[3] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong. Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing*, 33(3):46–54, 2011.