Efficient High Performance Computing by user networks (Extended Abstract)

Nunziato Cassavia[2], Sergio Flesca[1], Michele Ianni[1], Elio Masciari[2], Giuseppe Papuzzo[2], and Chiara Pulice[3]

[1] DIMES, University of Calabria, Italy
[2] ICAR-CNR, Italy
UMIACS, University of Maryland, USA
{flesca,mianni}@dimes.unical.it
{nunziato.cassavia,elio.masciari,giuseppe.papuzzo}@icar.cnr.it
{cpulice}@umiacs.umd.edu

Abstract. The advances in computational techniques both from a software and hardware viewpoint lead to the development of projects whose complexity could be quite challenging, e.g., biomedical simulations. In order to deal with the increased demand of computational power many collaborative approaches have been proposed in order to apply proper partitioning strategy able to assign pieces of execution to a crowd of workers. In this paper, we address this problem in a peer to peer way. We leverage the idling computational resources of users connected to a network. More in detail, we designed a framework that allows users to share their CPU and memory in a secure and efficient way. The latter allows users help each others by asking the network computational resources when they face high computing demanding tasks. As we do not require to power additional resources for solving tasks (we better exploit unused resources *already* powered instead), we hypothesize a remarkable side effect at steady state: energy consumption reduction compared with traditional server farm or cloud based executions.

1 Introduction

Computer Science is a really young discipline whose several branches grew up at impressive rates. Hardware cost is continuously decreasing while its performances are reaching unprecedented levels. Based on the availability of a plethora of powerful (portable) computing devices, new computing tasks have been designed and new data management paradigm emerged (e.g., the well known Big Data metaphore [8, 4, 1, 2]). Even though the advances in computer sciences lead to quite effective solution implementations, several problems still require too much computational resources for a single device execution. Indeed, since the introduction of Linux operating system, the idea of gathering from the crowd the resources for completing a task, have been widely leveraged.

The word Crowdsourcing was first introduced by Howe [5] in order to define the process of outsourcing some jobs to the crowd. It is used for a wide group of activities, as it allows companies to get substantial benefits by solving their problems in effective and efficient way. Indeed, crowd based solutions have been proposed for a wide range of applications as in image tagging [6], or sentiment analysis [3] to cite a few.

Many attempts have been made to properly define the very nature of collaborative distributed systems, however the solution to this apparently trivial task is far from being easy. As a matter of fact, all the successful systems (e.g., Wikipedia[3], Yahoo! Answers[4], Amazon Mechanical Turk[5]) rely on some assumptions: They should be able to involve project contributors, each contributor should solve a specific task, it is mandatory to effectively evaluate single contributions, and they should properly react to possible misconduct. The above mentioned points calls for a good trade-off between openness of the system and quality of service when designing a collaborative environment.

A well known open source framework that is widely used (mainly) for scientific purposes is BOINC (Berkeley Open Infrastructure for Network Computing)[6]. It allows volunteers to contribute to a wide variety of projects. The computing contribution is rewarded by credits for getting to a higher rank in a leaderboard. On the opposite side in recent years a new category of collaborative approaches is born for cryptovalue mining such as Bitcoin [7]. Users aiming at mining new Bitcoins contribute to a decoding task and are rewarded with a portion of the gathered money that is proportional to the effort put in the mining task. Our approach can be seen as a trade-off between BOINC framework implementation and Bitcoin mining.

More in detail, the novelty of our project is the exploitation of a peer to peer approach for reusing resources that users do not fully exploit to provide services that are currently offered by centralized server farm at an high cost with no customization. In particular, current limitation to high performance computing access (high price and difficult customization of resources), could be overcome by using our framework. Users who may need computing power, can simply ask other users that are not fully exploiting the potential of their computing devices (PC, smartphone, smart TV, etc.).

In order to assess the effectiveness of our solution, we analyzed a typical usage pattern for the big market of imaging and animation: the rendering of 3D scenes by a specialized plugin named *Mozaiko*^{TM} [7]. This operation typically engages users computers for several hours. Our experimental analyses evidence that existing solution are slower than our proposal due to their traditional approach to high performance computing and more expensive due to high prices of CPU cycles renting. Our plugin provides high added value to potential customers because it will be faster and cheaper than the competing solution (further details can be found at *www.coremuniti.com*).

Our solution do not require the continuous purchase of new servers, as users continuously provide (up to date) computational power. Finally, the better re-use of already powered resources could induce a beneficial systemic effect by reducing the overall

[3] https://www.wikipedia.org

[4] https://answers.yahoo.com/

[5] https://www.mturk.com

[6] https://boinc.berkeley.edu/

[7] We choose this name as Mozaiko is the Esperanto (a universal language) term denoting a Mosaic that is a good metaphore for our approach that partition a complex task in several sub-tasks that will be re-assembled like mosaic tiles

energy consumption for complex tasks execution. We plan to further investigate our conjecture in a future work as we are not able at this stage to generalize our early results.

Our Contribution. To summarize, we make the following major contributions: 1) We design and implement a hybrid P2P infrastructure that allows collaboration among users by sharing unexploited computational resources; 2) We define a robust model for task partitioning and assignment to network users;

2 Our system in a short

P2P networks features a common goal: the resources of many users and computers can be used in a collaborative way in order to significantly increase the computing power available for the users. In "full" P2P networks computers communicate directly each other thus allowing better bandwidth use. However, there are some inherent drawbacks to P2P solutions that require some functionalities to be centralized. Those systems can be used both for data sharing[10] and distributed computation[9], they are denoted as "hybrid" P2P. Our system, namely *Coremuniti*, falls in the latter category and aims to build a P2P network where users can share their unexploited computing resources. In Fig. 1 we sketched a possible usage scenario for our platform when using Mozaiko plugin. However, we point out that our platform is general purpose thus can be used for solving any complex problem that is parallelizable.

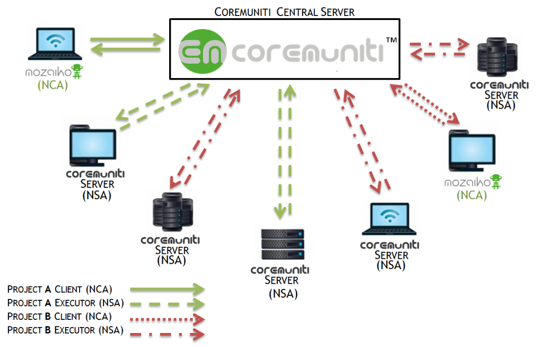

Fig. 1. Coremuniti system at work

Users who want to join our network, will download our software that will install the *Coremuniti server* on their devices (it is worth noticing that our software is platform independent). By running the node server, denoted in Fig. 1 as NSA (Node Server Agent), users can easily set the amount of CPU time they are willing to share with other users in the network. As they allow task execution, they will be rewarded by credits. We point out, that our software is executed in parallel with user own activities without any interaction with them. On the opposite side, users who need additional computing power to complete specialized tasks (in this paper we present the 3D Rendering case study) can install the specific software that is denoted in Fig. 1 as NCA that stands for Node Client Agent (i.e., *Mozaiko* for the rendering case study). As they start a new computing intensive task, they simply issue a request to the network for gathering the required resources. Two cases may occur: 1) they have previously gained (a portion of) the required credits for running the task (e.g., because they acted as servers) or 2) they have bought the required credits. In order, to guarantee a high level of service, we perform the subtask assignment step at the central server. More in detail, to fully take advantage from the capability of our network, we partition the (possibly huge) task in a suitable number of (much smaller) subtasks that can be quickly executed by the server peers.

As subtasks are completed, we check their correctness and reward the participating peers. In the following, we describe our model for subtask assignment that guarantees efficient execution for clients and gave to all peers (even if they have limited computing power) the possibility to be rewarded. Interesting enough, users that are not going to ask for task execution have the chance to accumulate credits that can be redeemed by coins or gadgets. As it will be shown in the experimental section, the latter features makes our framework a more convenient choice w.r.t to other collaborative systems such as cryptovalue miners. Indeed, in the early stage of development of our system we performed some preliminary user analysis by asking 500 students at University of Calabria their interest in joining the network. After specifying our rewarding model all of them agreed to join the network and they are currently beta testing the software. Due to space limitation, we will not describe the architecture details and in hat follows we will describe the most interesting part of our system, i.e. subtask assignment.

3 Subtask Assignment and Credit Rewarding

In this section, we describe our mathematical model for assigning subtask execution to resource providers whose objective is to minimize the expected completion time for the overall task. More in detail, as explained above, when a user asks the network for additional resources, the project s/he is going to execute is partitioned in several subtasks that can be completed using a limited amount of computational resources by every node connected to our P2P network . The obtained results are then combined for producing the project output much as much fast as possible than single machine execution.

The model assumes the presence of a set of available resource providers $\mathcal{RP} = \{rp_1, \ldots, rp_n\}$, and of a function $\lambda_c : \mathcal{RP} \rightarrow N \times N$, that assigns to every resource provider rp a pair $\langle tmin, tmax \rangle$, where $tmin$ (resp. $tmax$) is the minimum (resp. maximum) execution time required by the resource node to complete a subtask.

Furthermore, we leverage a credit assignment function which is based on the "usefulness" of the result yielded by the resource providers. Specifically, let st be a subtask whose execution was assigned c credits and which was assigned to the resource providers $rp_{i_1}, \ldots rp_{i_x}$. We order the resource providers $rp_{i_1}, \ldots rp_{i_x}$ ascending w.r.t. their completion times and store them in a sequence RP_{st}. Resource providers which did not terminate their subtask are put at the end of the sequence RP_{st} ordered according to the percentage of the subtask they completed.

When at least three resource providers completed their work, we assign $\frac{3c}{10}$ of the total credits, paid by the client for running the subtask on the network, to the first three resource providers in RP_{st}, i.e., the resource providers which "step up on the podium" for returning st result. Moreover, we distribute the remaining $\frac{c}{10}$ credits among the other resource provider in RP_{st} as follows. For each $j \in [4..x]$ we assign to $RP_{st}[j]$ the credits given by the following formula

$$\frac{c \cdot Compl(RP_{st}[j])}{10 \cdot \sum_{k=4}^{x} Compl(RP_{st}[k])},$$

where $Compl(rp)$ is the percentage of the subtask st which was completed by rp.

Tasks assignment aims at finding an assignment for resource provider to tasks which minimize the expected completion time while guaranteeing that:

1. every resource provider is assigned at most a subtask, and
2. every resource provider which is assigned a subtask have the possibility to be among the first three resource providers completing the subtask.

Subtask assignment proceeds in two phases. First an initial assignment of the various subtask to resource is yielded (Algorithm 1). Next, at regular intervals, the systems checks for the overall completion of the project and assigns resource providers that are available to new subtasks (Algorithm 2).

Let st be a subtask and $RP = \{rp_1, \ldots, rp_k\}$ be the set of resource providers assigned to st. The expected completion time of st given RP is denoted as $EC_{st,RP}$ and is computed as follows. Let $\boldsymbol{t}_{\downarrow}$ and $\boldsymbol{t}_{\uparrow}$ be two vectors reporting, respectively, the minimum and maximum completion times of the resource providers in RP in increasing order. Let t' denote the value $\boldsymbol{t}_{\downarrow}[\mathbf{2}]$ and t'' denote the value $\boldsymbol{t}_{\uparrow}[\mathbf{2}]$. Let $vect = [t_0 = t', \ldots, t_k = t'']$ be a vector containing all the values in $\boldsymbol{t}_{\downarrow}$ and $\boldsymbol{t}_{\uparrow}$ which lie in the interval $[t', t'']$. Hence, denoting with $F^3(x)$ the probability that at least three resource providers complete their job before time limit x, we have that:

$$\begin{aligned} EC_{st,RP} = \textstyle\int_0^{\infty}(1 - F^3(x))dx = t' + \int_{t'}^{t''}(1 - F^3(x))dx = \\ t' + \textstyle\sum_{i=0}^{k-1} \int_{t_i}^{t_{i+1}}(1 - F_i^3(x))dx \ = \\ t' + \textstyle\sum_{i=0}^{k-1} FF_i^3(t_i, t_{i+1}), \end{aligned}$$

where $F_i^3(x)$ is the probability that at least three resource providers complete their work before x time limit given that $t_i \leq x \leq t_{i+1}$ and $FF_i^3(t_i, t_{i+1})$ is equal to $\int_{t_i}^{t_{i+1}}(1 - F_i^3(x))dx$. It is easy to see that $FF_i^3(t_i, t_{i+1})$ can be easy derived form the minum and maximum completion times associated to every resource providers by the function λ_c.

Example 1. Consider the case that a set of four resource providers $RP = \{rp_1, \ldots, rp_4\}$ was assigned to st where $\lambda_c(rp_1) = \langle 1, 5\rangle, \lambda_c(rp_2) = \langle 2, 7\rangle, \lambda_c(rp_3) = \langle 6, 7\rangle$ and

$\lambda_c(rp_4) = \langle 4, 8 \rangle$. In this case, $t' = 4$ and $t'' = 7$ and

$$
\begin{gathered}
EC_{st,RP} = \\
4 + \int_4^5 (1 - F_0^3(x))dx + \int_5^6 (1 - F_1^3(x))dx + \int_6^7 (1 - F_2^3(x))dx = \\
4 + \int_4^5 (1 - \tfrac{x-1}{5-1}\tfrac{x-2}{7-2}\tfrac{x-4}{8-4})dx + \int_5^6 (1 - \tfrac{x-2}{7-2}\tfrac{x-4}{8-4})dx + \\
\int_6^7 \Big(1 - \Big(\tfrac{x-2}{7-2}\tfrac{x-6}{7-6} + \tfrac{x-2}{7-2}\left(1 - \tfrac{x-6}{7-6}\right)\tfrac{x-4}{8-4} + \left(1 - \tfrac{x-2}{7-2}\right)\tfrac{x-6}{7-6}\tfrac{x-4}{8-4}\Big)\Big)\, dx = \\
4 + \int_4^5 (1 - \tfrac{x-1}{4}\tfrac{x-2}{5}\tfrac{x-4}{4})dx + \int_5^6 (1 - \tfrac{x-2}{5}\tfrac{x-4}{4})dx + \\
\int_6^7 (1 - (\tfrac{x-2}{5}\tfrac{x-6}{1} + \tfrac{x-2}{5}(1 - \tfrac{x-6}{1})\tfrac{x-4}{4} + (1 - \tfrac{x-2}{5})\tfrac{x-6}{1}\tfrac{x-4}{4}))\, dx = \\
4 + \left[\tfrac{11x}{10} - \tfrac{7x^2}{80} + \tfrac{7x^3}{240} - \tfrac{x^4}{320}\right]_4^5 + \left[\tfrac{3x}{5} + \tfrac{3x^2}{20} - \tfrac{x^3}{60}\right]_5^6 \\
+ \left[\tfrac{1}{10}(-126x + 44x^2 - \tfrac{17x^3}{3} + \tfrac{x^4}{4})\right]_6^7 = \\
4 + \tfrac{901}{960} + \tfrac{11}{15} + \tfrac{31}{120} \approx 5.93
\end{gathered}
$$

The initial subtask assignment is computed by running Algorithm 1. It works in two phases. First an initial assignment of resource providers to tasks is yielded by assigning a resource provider at a time to the subtask which minimize the maximum expected completion time of all the tasks, considering resource providers in ascending order of their expected completion time (lines 2-8). Next the initial assignment is revised by swapping pairs of resource providers between tasks. The algorithm iteratively selects a pair of assignments $\langle st', rp' \rangle, \langle st'', rp'' \rangle$ which once swapped provide the greatest decrease of the maximum expected completion time (lines 9-12). Finally, the subtask assignment is returned.

Algorithm 1 Initial Assignment

```
Input: Resource provider sequence $\mathcal{RP}$ of size $n$
Input: Subtask sequence $\mathcal{ST}$ of size $k < \frac{n}{3}$
Output: Tasks assignment $\mathcal{SA}$
 1: $\mathcal{SA} = \emptyset$
 2: $\mathcal{RP} = orderOnExpCompl(\mathcal{RP})$
 3: for $i = 1$ to $n$ do
 4:     $st = selectBestFreeST(\mathcal{ST}, \mathcal{SA}, \mathcal{RP}[i])$
 5:     if $st \neq \texttt{null}$ then
 6:         $\mathcal{SA} = \mathcal{SA} \cup \langle st, \mathcal{RP}[i] \rangle$
 7:     end if
 8: end for
 9: repeat
10:     $(\langle st', rp' \rangle, \langle st'', rp'' \rangle) = selectBestCandidatePair(\mathcal{SA})$
11:     $\mathcal{SA} = \mathcal{SA} - \{\langle st', rp' \rangle, \langle st'', rp'' \rangle\} \cup \{\langle st', rp'' \rangle, \langle st'', rp' \rangle\}$
12: until $existsACandidatePair(\mathcal{SA})$
13: return $\mathcal{SA}$
```

Function *orderOnExpCompl* takes as input a sequence of resource providers $\mathcal{RP}$ and returns $\mathcal{RP}$ ordered ascending w.r.t. the expected resource provider completion time. Function *selectBestFreeST* returns, given a resource provider rp and a subtask assignemnt $\mathcal{SA}$, the subtask $st \in ST$ such that $MaxECP(\mathcal{SA} \cup \{\langle st, rp\rangle\})$ is the minimum, i.e., $st = \arg\min_{st \in \mathcal{ST}}(MaxECP(\mathcal{SA} \cup \{\langle st, rp\rangle\}))$, such that $MaxECP(\mathcal{SA} \cup \{\langle st, rp\rangle\}) < MaxECP(\mathcal{SA})$, `null` otherwise. Since in the initial step of the algorithm it may happen that there is some subtask st in $\mathcal{ST}$ such that $|\mathcal{SA}(st)| = x < 3$, for each subtask st satisfying this constraint, when computing $MaxECT$ function *selectBestFreeST* assumes that $3 - x$ fake resource providers have been assigned to st, where the minimum and maximum completion times of these fake resource providers are $max - 1$ and max, respectively, where max is a constant value (much) greater than the maximum completion time of every the resource provider in $\mathcal{RP}$.

Function *selectBestCandidatePair* returns a pair of assignments $\langle st, rp\rangle$ and $\langle st', rp'\rangle$ in $\mathcal{SA}$ such that there not exist assignment $\langle st'', rp''\rangle$ and $\langle st^*, rp^*\rangle$ in $\mathcal{SA}$ such that $MaxECP(\mathcal{SA} - \{\langle st, rp\rangle, \langle st', rp'\rangle\} \cup \{\langle st, rp'\rangle, \langle st', rp\rangle\}) > MaxECP(\mathcal{SA} - \{\langle st'', rp''\rangle, \langle st^*, rp^*\rangle\} \cup \{\langle st'', rp^*\rangle, \langle st^*, rp''\rangle\})$. Function *existsACandidatePair* returns true if there is a pair of assignments $\langle st, rp\rangle$ and $\langle st', rp'\rangle$ in $\mathcal{SA}$ such that $MaxECP(\mathcal{SA} - \{\langle st, rp\rangle, \langle st', rp'\rangle\} \cup \{\langle st, rp'\rangle, \langle st', rp\rangle\}) < MaxECP(\mathcal{SA})$. Moreover, both functions *selectBestCandidatePair* and *existsACandidatePair* consider only pairs of assignments $\langle st, rp\rangle$ and $\langle st', rp'\rangle$ in $\mathcal{SA}$ such that swapping rp and rp' guarantees that the probability that rp (resp. rp') is among the first three resource providers assigned to st' (resp. st) that complete st is greater than zero.

Our central server runs Algorithm 2 at fixed intervals, in order to check for the overall completion of the project and assigns resource providers, that have completed the assigned tasks, to further tasks whose results are not yet available. Before running this algorithm the system updates the statistics of resource providers which are assigned to a subtask yielding up to date values for the minimum and maximum completion times. Essentially Algorithm 2 first orders the available resource providers ascending w.r.t. their expected completion times and next iteratively assigns each resource provider rp to the subtask st selected by invoking function *selectBest*. Function *selectBest* returns, given a resource provider rp and a subtask assignemnt $\mathcal{SA}$, the subtask $st \in ST$ such that $MaxECP(\mathcal{SA} \cup \langle st, rp\rangle)$ is the minimum, i.e., $st = \arg\min_{st \in \mathcal{ST}}(MaxECP(\mathcal{SA} \cup \langle st, rp\rangle))$, and is such that $MaxECP(\mathcal{SA} \cup \{\langle st, rp\rangle\}) < MaxECP(\mathcal{SA})$, `null` otherwise.

4 Conclusion and Future Work

In this paper, we proposed a hybrid peer to peer architecture for computational resource sharing. By our network users can provide their unexploited computational resource to those users who run computational demanding tasks and they are rewarded for their collaboration. In order to guarantee the efficiency and effectiveness of the computation process, we design a task partitioning and assignment algorithm that reduce the execution times while allowing satisfactory revenues for resource providers.

Acknowledgment

The authors thank the Calabria region that fully funded the project development and the EU commission that awarded our project with a seal of excellence.

Algorithm 2 Incremental Assignment

```
Input: Available resource providers sequence $\mathcal{RP}$ of size $n$
Input: Subtask sequence $\mathcal{ST}$
Input: Initial Subtasks assignment $\mathcal{SA}$
Input: Subtasks and Resource Providers constraints $STC$
Output: Revised Subtasks assignment $\mathcal{SA}$
1: $\mathcal{RP} = orderOnExpCompl(\mathcal{RP})$
2: for $i = 1$ to $n$ do
3:     $st = selectBest(\mathcal{ST}, \mathcal{SA}, \mathcal{RP}[i])$
4:     if $st \neq$ null then
5:         $\mathcal{SA} = \mathcal{SA} \cup \langle st, \mathcal{RP}[i] \rangle$
6:     end if
7: end for
8: return $\mathcal{SA}$
```

References

1. D. Agrawal et al. Challenges and opportunities with big data. A community white paper developed by leading researchers across the United States. 2012.
2. V. R. Borkar, M. J. Carey, and C. Li. Inside "Big Data Management": Ogres, Onions, or Parfaits? In *International Conference on Extending Database Technology*, pages 3–14, 2012.
3. A. Brew, D. Greene, and P. Cunningham. Using crowdsourcing and active learning to track sentiment in online media. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 145–150, 2010.
4. T. Economist. Data, data everywhere. *The Economist*, Feb 2010.
5. J. Howe. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*. Crown Publishing Group, New York, NY, USA, 1 edition, 2008.
6. X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: A crowdsourcing data analytics system. *Proc. VLDB Endow.*, 5(10):1040–1051, June 2012.
7. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Freely available on the web*, 2008.
8. Nature. Big data. *Nature*, Sept. 2008.
9. B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 561–570, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
10. M. Yang and Y. Yang. An efficient hybrid peer-to-peer system for distributed data sharing. *IEEE Transaction on Computer*, 59(9):1158–1171, 2010.