# CloudFIT, a PaaS platform for IoT applications over Pervasive Networks

Luiz Angelo Steffenel[1] and Manuele Kirch Pinheiro[2]

[1] CReSTIC Laboratory, SysCom team
Universit de Reims Champagne-Ardenne
Reims, France
luiz-angelo.steffenel@univ-reims.fr
[2] Centre de Recherche en Informatique
Universite Paris 1 Panthon-Sorbonne
Paris, France
manuele.kirsch-pinheiro@univ-paris1.fr

**Abstract.** IoT applications are the next important step towards the establishment of mobiquitous systems, but at the same time these environments raise important challenges when considering data distribution and processing. While most IoT applications today rely on clouds as back-end, critical applications that require fast response or enhanced privacy levels may require proximity services specially tailored to these needs. The deployment of private cloud services on top of pervasive grids represent an interesting alternative to traditional cloud infrastructures. In this work we present CloudFIT, a PaaS middleware that allows the creation of private clouds over pervasive environments. Using a Map-Reduce application as example, we show how CloudFIT provides both storage and data aggregation/analysis capabilities at the service of IoT networks.

## 1 Introduction

Today, cloud computing is a widespread paradigm that relies on the externalization of services to a distant platform with elastic computing capabilities. Unsurprisingly, Big Data analytics profits from the computing capabilities from the cloud, making it the predilection platform for information extraction and analysis.

The emergence of Internet of Things (IoT) has naturally attired the attention of developers and companies, which mostly rely on cloud services to interconnect devices and gather information. Indeed, platforms like Carriots [3] or ThinkSpeak [4] now propose PaaS APIs to collect information, visualize and control IoT devices.

Contrarily to the case of Wireless Sensor Networks (WSNs), however, IoT has a much more complex data transfer pattern that is not always tailored for a cloud. While data from WSNs naturally flows from the sensors to a ”sink” repository that can gather information and handle it to the analytics software, IoT devices have M2M (Machine-to-Machine) capabilities beyond simple raw data transmission, as they are also information consumers and even actuators to the real environment.

[3] https://www.carriots.com/
[4] https://thingspeak.com/

Simply relying on a distant cloud infrastructure for data storage, processing and control imposes a non-negligible latency, a complete dependency on wide-area communications and the transmission of potentially sensible data across the network. From this point of view, it is clear that not all IoT applications would benefit from an external data handling.

Deploying a privative PaaS cloud for IoT is an interesting alternative to the complete externalization, as it ensures fast reaction and privacy levels tailored to the specific needs of an enterprise or application. Indeed, the omnipresence of IoT devices often raises questions about the dissemination of sensitive data, a problem that public cloud systems can minimize through the use of heavy layers of cryptography and anonymization, but never solve.

In this paper we present CloudFIT, a distributed computing middleware designed for pervasive environments that offers IoT applications both storage, data aggregation and analysis capabilities. In addition, CloudFIT does not require a dedicated infrastructure as a CloudFIT ”grid” can be deployed over existing resources on the enterprise (desktop PCs, servers, etc.) and perform both the data aggregation, filtering and analysis required by IoT devices.

After describing CloudFIT, we illustrate its operation through the deployment of a data intensive application over a cluster. We deploy a MapReduce application over CloudFIT, and compare its performance against the well-known Hadoop middleware[5], a Big Data platform specially designed for dedicated clusters.

The paper is structured as follows: Section 2 discusses the challenges for the IoT applications and the reasons why a traditional cloud services is not always recommended. Instead, we emphasize alternatives for cloud computing that ensure both efficiency and data privacy. Section 3 focuses on the case of data-intensive problems and discusses the main challenges for its deployment over pervasive grids, analyzing some related works. Section 4 presents the architecture of CloudFIT and its characteristics related to fault tolerance and volatility support. This session also discuss how to interface IoT devices and applications with CloudFIT. Section 5 introduces our implementation of a MapReduce application over CloudFIT, discussing both implementation issues and performance evaluations. Finally, Section 6 concludes this paper and sets the lines of our next development efforts.

## 2 Cloud services and IoT

### 2.1 Private Clouds, Cloudlets and the IoT

When the cloud computing paradigm started, we observed the development of middlewares and tools for the establishment of private and mixed cloud infrastructures. Most of these tools, like Eucalyptus [18], Nimbus [12] or OpenNebula [17], are designed to provide IaaS on top of dedicated resources like clusters or private datacenters. While extremely powerful, the deployment of these environments is complex and requires dedicated resources, which minimizes their advantage against public cloud infrastructures like Amazon EC2.

[5] `http://hadoop.apache.org/`

Establishing on-demand cloud services on top of existing resources is also alternative to the complete externalization of services in a cloud. For example, [22,23] explore the limitations of mobile devices and the inaptitude of current solutions to externalize mobile services through the use of Cloudlets, i.e., virtual machines deployed on-demand in the vicinity of the demanding devices. Using cloudlets deployed as Wi-Fi hotspots in coffee shops, libraries, etc., the authors of [22,23] suggest a simple way to offer enough computing power to perform complex computations (services) all while limiting the service latency. Please note that these cloudlets do not work as a single entity/platform but instead act as detached handlers for specific demands.

Proximity cloud services can also be used to perform an initial processing on the data. For instance, [20] presents a platform where context information is collected, filtered and analyzed on several layers. This way, basic context actions may be decided/performed in a close area range, while a much deep analysis of the context information may be performed by external servers. This layered analysis can also be used to ensure privacy properties, for example by anonymizing the data that will be used to the global context analysis. As context my represent multiple and heterogeneous kind of information, this approach can also be implemented to general Big Data analytics on sensor data or access logs, for example.

Another usage for private clouds relates to the reinforcement of the security of a network [11]. In a mobile network (as well as in an IoT pervasive network), devices cannot rely in a single security device in the entrance of the network because multimodal connections may be established with outside devices via Wi-Fi, 3G, Bluetooth, etc. If nowadays similar procedures can be implemented through the use of 802.1x authentication or VPNs, their configuration complexity requires a high technical knowledge. A better alternative relies on a mutual monitoring system sharing information is created around a confidence zone (community). Joining a confidence zone is only possible if the device pass some control checks and, similarly, devices that become ”dangerous due to a virus or a Trojan can be blocked and removed from the community.

We consider that deploying cloud services for IoT over pervasive networks is a natural approach, as the heterogeneity and the dynamicity of the devices impose a frequent adaptation on both network interconnections and computing requirements.

### 2.2 Cloud services over Pervasive Grids

Pervasive grids can be defined as large-scale infrastructures with specific characteristics in terms of volatility, reliability, connectivity, security, etc. According to [19], pervasive grids represent the extreme generalization of the grid concept, seamlessly integrating pervasive sensing/actuating instruments and devices together with classical high performance systems. In the general case, pervasive grids rely on volatile resources that may appear and disappear from the grid, according their availability. Indeed, mobile devices should be able to come into the environment in a natural way as their owner moves [6]. Also, devices from different natures, from the desktop and laptop PCs until the last generation tablets, should be integrated in seamlessly way. These environments are therefore characterized by three main requirements:

– The volatility of its components, whose participation is a matter of opportunity and availability;

– The heterogeneity of these components, whose capabilities may vary on different aspects (platform, OS, memory and storage capacity, network connection, etc.);
– The dynamic management of available resources, since the internal status of these devices may vary during their participation into the grid environment.

Such dynamic nature of pervasive grids represents an important challenge for executing data intensive applications. Context-awareness and nodes volatility become key aspects for successfully executing such applications over pervasive grids, but also for the handling and transmission of voluminous datasets.

Our approach to implement cloud-like services over pervasive networks relies on the use of an overlay network provided by a P2P system. In this approach, the P2P overlay provides all communication and fault tolerance properties required for the operation on a pervasive network, as well as some additional services like DHT storage that can help implementing additional services.

Indeed, if P2P systems are widely known for their use on storage and sharing applications, they can also be used as platforms for coordination and distribution of computing tasks. Solutions like CONFIIT [10], DIET [3] have demonstrated the interest of P2P to support computing problems in distributed and heterogeneous environments.

## 3 Data-intensive applications on Pervasive Grids

In spite of a wide tradition on distributed computing projects, most pervasive grid middlewares have focused on computing-intensive parallel applications with few I/O and loose dependencies between the tasks. Enabling these environments to support data-intense applications is still a challenge, both in performance and reliability. We believe that MapReduce is an interesting paradigm for data-intensive applications on pervasive grids as it presents a simple task distribution mechanism, easily implemented on a pervasive environment, but also a challenging data distribution pattern. Enabling MapReduce on pervasive grids raises many research issues, which we can decompose in two subtopics: data distribution and data processing.

There are two approaches to distribute large volume of data to large number of distributed nodes. The first approach relies on P2P protocols where peers collaboratively participate to the distribution of the data by exchanging file chunks [27,7,15]. The second approach is to use a content delivery service where files are distributed to a network of volunteers [16,13].

Concerning data processing on pervasive grids, some authors have tried to improve the processing capabilities of Hadoop to take into account the volatility of the nodes. Indeed, Zaharia et al. [28] Chen et al. [5] or Ahmad et al. [1] deals with heterogeneity of the supporting infrastructure, proposing different scheduling algorithms that can improve Hadoop response time. Lin et al. [14] explore the limitations of Hadoop over volatile, non-dedicated resources. They propose the use of a hybrid architecture where a small set of reliable nodes are used to provide resources to volatile nodes.

Due to the simplicity of its processing model (map and reduce phases), data processing can be easily adapted to a given distributed middleware, which can coordinate tasks through different techniques (centralized task server, work-stealing/bag of tasks, speculative execution, etc.). Nevertheless, good performances can only be achieved through the minimization of data transfers over the network, which is one of the key aspects of Hadoop HDFS filesystem. Only few initiatives associate data-intense computing with large-scale distributed storage on volatile resources. In [4], the authors present an architecture following the super-peer approach where the super-peers serve as cache data server, handle jobs submissions and coordinate execution of parallel computations.

## 4 CloudFIT

In this work we present our efforts to enable MapReduce applications over the P2P distributed computing middleware CloudFIT [24,26]. The CloudFIT framework (Figure 1) is structured around collaborative nodes connected over an overlay network. CloudFIT was designed to be independent of the underlying overlay, and the current version supports both Pastry [21] and TomP2P overlay networks [2]. Pastry is one of the most known P2P overlays and is widely employed in distributed computing environments. TomP2P is a more recent P2P library, enjoying an active development community.

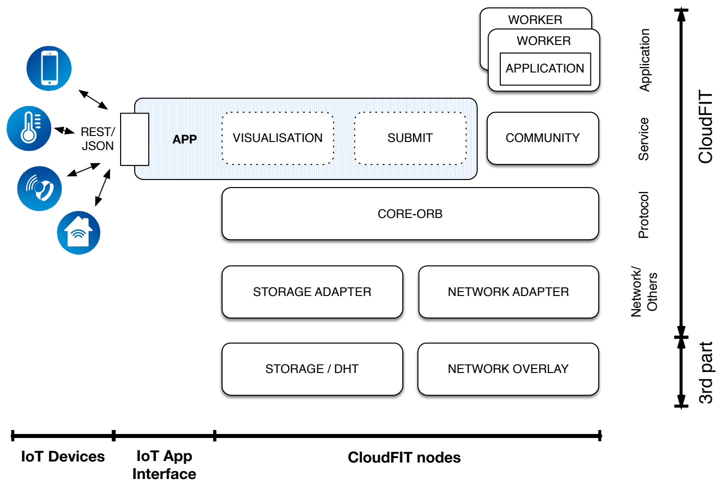

**Fig. 1.** CloudFIT architecture stack

An application for CloudFIT must provide a java class that implements two basic API methods: how many tasks to solve (`setNumberOfBlocks()`) and how to compute an individual task (`executeBlock(number, required[])`). When executing, each node owns the different parameters of the current computations (a list of tasks and associated results) and is able to locally decide which tasks still need to be computed and can carry the work autonomously if no other node can be contacted. Access to the storage is also provided through the API, if required. The status of completed tasks (optionally including the partial results from these tasks) are distributed among the nodes, contributing therefore to the coordination of the computing tasks and form a global view of the calculus.

The basic scheduling mechanism simply randomly rearranges the list of tasks at each node, which helps the computation of tasks in parallel without requiring additional communication between nodes. This simple scheduler mechanism was designed to allow idle processes to speculatively execute incomplete tasks, reducing the ”tail effect” when a task is computed by a slow node. The scheduling mechanism supports task dependencies (allowing the composition of DAGs) and can be also be improved through the use of a context module [25] that provides additional information about the nodes capacities.

Finally, fault tolerance is ensured both by the overlay (network connections, etc.) and by the computing platform. Indeed, as long as a task is not completed, other nodes on the grid may pick it up for execution. In this way, when a node fails or leaves the grid, other nodes may recover tasks originally started by the crashed node. Inversely, when a node joins the CloudFIT community, it receives an update about the tasks current status and the working data, allowing it to start working on available (incomplete) tasks.

### 4.1 CloudFIT Services for IoT Devices and Applications

As previously stated, CloudFIT provides a pervasive PaaS for IoT applications. While we believe that CloudFIT can be deployed directly over IoT devices running Android (with the TomP2P overlay) or Linux on Raspberry Pi, the heterogeneity and limited resources of these devices make this approach very unreliable. Indeed, a node integrating the CloudFIT network must perform all the routing, storage and computing tasks as the others, and this can be both overloading and inneficient (please see Section 5.5).

A better approach, instead, is to use CloudFIT as a computing backend for IoT devices and applications. This mixed architecture, as illustrated in the left side of Figure 1, allows an IoT application connected to CloudFIT network to act as an interface to gather data and launch computing tasks according to the application needs.

While the development of an interface for IoT devices can be provided through *REST/json* calls or even a direct a connection to the devices via BlueTooth or Wi-Fi, it is outside the scope of this paper. Instead, the next sections illustrate the deployment of a MapReduce application over CloudFIT. This is one of several computing intensive tasks that can be performed on CloudFIT to support IoT applications.

## 5 MapReduce over CloudFIT

### 5.1 MapReduce

MapReduce [8] is a parallel programming paradigm successfully used by large Internet service providers to perform computations on massive amounts of data. The key strength of the MapReduce model is its inherently high degree of parallelism that should enable processing of petabytes of data in a couple of hours on large clusters.

Computations on MapReduce deal with pairs of key-values $(k, V)$, and a MapReduce algorithm (a job) follows a two-step procedure:

1. map: from a set of key/value pairs from the input, the map function generates a set of intermediate pairs $(k_1; V_1) \rightarrow \{(k_2; V_2)\}$;

2. reduce: from the set of intermediate pairs, the reduce function merges all intermediate values associated with the same intermediate key, so that $(k_2; \{V_2\}) \rightarrow \{(k_3; V_3)\}$.

When implemented on a distributed system, the intermediate pairs for a given key $k_2$ may be scattered among several nodes. The implementation must therefore gather all pairs for each key $k_2$ so that the reduce function can merge them into the final result. Additional features that may be granted by the MapReduce implementation include the splitting of the input data among the nodes, the scheduling of the jobs' component tasks, and the recovery of tasks hold by failed nodes.

Hadoop, one of the most popular implementations of MapReduce, provides these services through a dual layered architecture where tasks scheduling and monitoring are accomplished through a master-slave platform, while the data management is accomplished by a second master-slave platform on top of the hierarchical HDFS file-system. Such master-slave architecture makes Hadoop not suitable for Pervasive Grids.

### 5.2 Map, Reduce and task dependencies

In order to implement a MapReduce application under the FIIT model, tasks inside a Map or Reduce job must be independent, all while preserving a causal relation between Map and Reduce. Therefore, several tasks are launched during the Map phase, producing a set of $(k_i, V_i)$ pairs. Each task is assigned to a single file/data block and therefore may execute independently from the other tasks in the same phase. Once completed, the results from each task can be broadcasted to all computing nodes and, by consequence, each node contains a copy of the entire set of $(k_i, V_i)$ pairs at the end of the Map phase. At the end of the first step, a Reduce job is launched using as input parameter the results from the map phase.

In our prototype, the number of Map and Reduce tasks was defined to roughly mimic the behavior of Hadoop, which tries to guess the required number of Map and Reduce processes. For instance, we set the number of Map tasks to correspond to the number of input files, and the number of Reduce tasks depends on the size of the dataset and the transitive nature of the data. Please note that CloudFIT may optionally perform a result aggregation after each job completion, just like Hadoop *combiners*.

Because Hadoop relies on specific classes to handle data, we tried to use the same ones in CloudFIT implementation as a way to keep compatibility with the Hadoop API. However, some of these classes were too dependent on inner elements of Hadoop, forcing us to develop our own equivalents, at least for the moment (further works shall reinforce the compatibility with Hadoop API). For instance, we had to substitute the OutputCollector class with our own MultiMap class, while the rest of the application remains compatible with both Hadoop and CloudFIT.

### 5.3 Data management, storage and reliability

As stated before, CloudFIT was designed to broadcast the status about completed tasks to all computing nodes, and this status may include the tasks' results. By including the results, CloudFIT ensures $n - resiliency$ as all nodes will have a copy of the data.

This resiliency behavior was mainly designed for computing intensive tasks that produce a small amount of data as result. On data-intensive applications, however, $n-resiliency$ may be prohibitive as not only all nodes need to hold a copy of all task's data, but also because broadcasting several megabytes/gigabytes over the network is a major performance issue.

In our efforts to implement MapReduce over CloudFIT we chose a different approach to ensure the scalability of the network all while preserving good reliability levels. Hence, we rely on the DHT to perform the storage of tasks results as {*task_key, task_result*} tuples, while the task status messages broadcast the keys from each task. As both PAST and TomP2P DHT implement data replication among the nodes with a predefined replication factor $k$, we can ensure minimal fault tolerance levels all while improving the storage performance.

### 5.4 Performance evaluation against Hadoop

In order to evaluate the performance of MapReduce over CloudFIT we implemented the traditional WordCount application and compared it against WordCount 1.0 application from Hadoop tutorial.

To make this first evaluation fair, we conducted this first experiment over 8 machines from the ROMEO Computing Center[6]. ROMEO cluster nodes are composed by bi-Intel Xeon E5-2650 2.6 GHz (Ivy Bridge) 8 cores and 32GB of memory, interconnected by an Infiniband QDR network at 40Gbps. Hadoop YARN nodes run with default parameters (number of *vcores* = 8, available memory = 8GB), parameters that we reproduced on CloudFIT for fairness (i.e., by limiting the number of parallel tasks by node and setting the maximum java VM memory).

Two different versions of CloudFIT were tested, one using the FreePastry overlay with the PAST DHT at the storage layer, and the second one with the TomP2P overlay and its Kademlia-based DHT.

The experiments considered the overall execution time (map + reduce phases) of both CloudFIT and Hadoop implementations when varying the total amount of data (512MB to 2GB). The data was obtained from a corpus of textbooks from the Gutenberg Project and split in blocks of 64MB to reproduce the size of an HDFS data block. The results obtained when running on an 8 nodes cluster are presented on Figure 2, which shows the average of 10 executions for each data size.

At first glance, we observe that the CloudFIT/TomP2P implementation easily outperforms both CloudFIT/PAST and Hadoop, which have aproximately the same performance. A deeper analysis of the CloudFIT/PAST implementation show that the PAST DHT experimented a performance bottleneck related to the use of mutable objects. Indeed, mutable objects are useful to gather $(k;V)$ pairs from different tasks but they force a non-negligible overhead at the DHT controller, which must scan the data for changes and trigger replication updates. One solution to improve the PAST performance is to rely on immutable objects that do not suffer from this problem, but this requires the usage of alternative data structures to reproduce the $(k;V)$ associations from MapReduce.

[6] `https://romeo.univ-reims.fr`

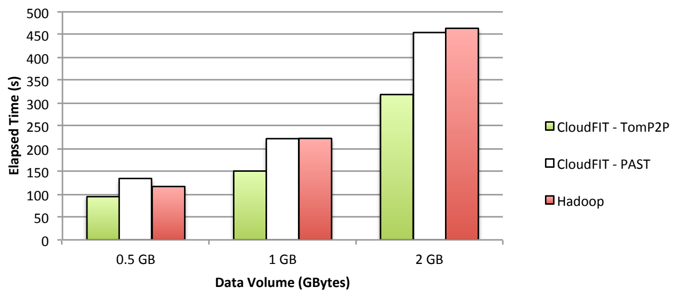

**Fig. 2.** WordCount MapReduce performance on 8 nodes

This is an encouraging result as it demonstrates the interest of CloudFIT as a platform for Big Data applications. Depending on the storage layer, we can provide good performance levels without sacrificing the platform flexibility. In addition, the modular organization of CloudFIT allows connecting other storage supports like BitDew [9], external databases, URLs, etc., according to the application requirements.

## 5.5 Performance evaluation on a pervasive grid

As the previous section demonstrate that CloudFIT can execute MapReduce applications as fast as Hadoop in a HPC cluster, the next step in our experiments considered the creation of a pervasive cluster on top of common desktop equipments. For instance, we executed CloudFIT/TomP2P over a network composed by three laptop computers connected through a Wi-Fi 802.11g network. The specifications for each model are presented in Table 1. Please note that these machines were not tuned for performance and indeed CloudFIT had to share the resources with other applications like anti-virus, word processors, etc.

| Laptop | Processor | GHz | Cores | Threads | Memory | OS |
|---|---|---|---|---|---|---|
| MacBook Air | Intel® Core™ i7-4650U | 1.7 | 2 | 4 | 8 GB | MacOS 10.10.3 |
| HP Pavillon dv6 | Intel® Core™ i5-2450M | 2.5 | 2 | 4 | 8 GB | Windows 7 |
| Lenovo U110 | Intel® Core™2 Duo L7500 | 1.6 | 2 | 2 | 4 GB | Ubuntu Linux 15.4 |

**Table 1.** Specification of the nodes on the pervasive cluster

Figure 3 presents the execution of the WordCount application with three different data sets, and we compare the execution time obtained on the pervasive grid with the performance obtained over 3 nodes from the ROMEO cluster. Post-execution analysis indicated that in spite of the processors type and speeds, one factor that mainly influenced the performance was the network speed. Indeed, as the MapReduce application performs several read/write operation over the DHT, the network is a major bottleneck: to write 64MB of data on the DHT using the ROMEO cluster (equipped with an Infiniband interconnection) we need in average 2 seconds, while the Wi-Fi connection used on the pervasive cluster required in average 15 seconds. Another element that contributes to the reduced performance of the pervasive environment is the competition between faster and slower nodes: while both node types have similar chances to draw tasks to execute at the beginning, faster nodes will complete their tasks first and finally re-execute the tasks from slower nodes, wasting computing resources.

While comparing both environments is not really fair, the conclusion is that one does need a dedicated environment to extract enough computing power for several applications. In fact, the flexibility of the pervasive cluster allows nodes to join or leave the cluster without interfering with the execution, making it a strategic tool for most organizations that cannot rely neither in a dedicated cluster neither in a distant data-center/cloud infrastructure. Further, CloudFIT has the advantage that it can be easily run on Windows, contrarily to Hadoop, which reinforcing its ability to create pervasive clusters from the available resources.

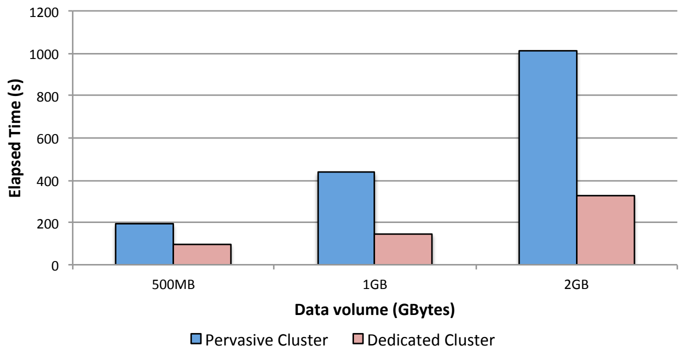

**Fig. 3.** WordCount MapReduce on 3 nodes: pervasive vs dedicated cluster

## 6 Conclusions and Future Work

IoT networks are the next important step towards the establishment of mobiquitous systems. Contrarily to Sensor Networks, IoT has a much richer M2M pattern that is not always adapted to the cloud computing paradigm. Indeed, moving data to distant platforms for filtering, analysis and decision-making is both expensive and time consuming, which not always fits the IoT applications requirements.

In this paper we present CloudFIT, a PaaS middleware that allows the creation of private clouds at the proximity of the demanding IoT devices. Using a P2P overlay, CloudFIT offers both storage and computing capabilities on top of pervasive networks.

We illustrate the usage of CloudFIT through the deployment of a MapReduce application and the comparative performance analysis with Hadoop. Indeed, we demonstrate that CloudFIT offers performance levels similar to those of Hadoop but with a better support for dynamic and heterogeneous environments.

Of course, the possibilities that CloudFIT offers to IoT are not limited to MapReduce applications. The CloudFIT API and its distributed computing model allow many other usages, as devices can use the platform as a storage support, data analysis support, intensive computing support, etc. By coordinating activities over CloudFIT, IoT devices and applications can elaborate a supply chain from data gathering to reasoning and actuation.

## Acknowledgment

The authors would like to thank their partners in the PER-MARE project (`http://cosy.univ-reims.fr/PER-MARE`) and acknowledge the financial support given to this research by the CAPES/MAEE/ANII STIC-AmSud collaboration program (project number 13STIC07).

## References

1. Ahmad, F., Chakradhar, S.T., Raghunathan, A., Vijaykumar, T.N.: Tarazu: optimizing mapreduce on heterogeneous clusters. SIGARCH Comput. Archit. News 40(1), 61–74 (Mar 2012)
2. et al., T.B.: Tomp2p, a p2p-based high performance keyvaluepair storage library, `http://tomp2p.net/`
3. Caron, E., Desprez, F., Lombard, F., Nicod, J.M., Quinson, M., Suter, F.: A scalable approach to network enabled servers. In: Monien, B., Feldmann, R. (eds.) Proceedings of the 8th International EuroPar Conference. Lecture Notes in Computer Science, vol. 2400, pp. 907–910. Springer-Verlag, Paderborn, Germany (Aug 2002)
4. Cesario, E., Mastroianni, C., De Caria, N., Talia, D.: Distributed data mining using a public resource computing framework. Grids, P2P and Service Computing (2010)
5. Chen, Q., Zhang, D., Guo, M., Deng, Q., Guo, S.: Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In: Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology. pp. 2736–2743. CIT '10, IEEE Computer Society, Washington, DC, USA (2010)
6. Coronato, A., Pietro, G.D.: Mipeg: A middleware infrastructure for pervasive grids. Future Generation Computer Systems 24(1), 17 – 29 (2008)
7. Costa, F., Silva, L., Fedak, G., Kelley, I.: Optimizing data distribution in desktop grid platforms. Parallel Processing Letters 18(3), 391–410 (Sep 2008)
8. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)
9. Fedak, G., He, H., Cappello, F.: Bitdew: a programmable environment for large-scale data management and distribution. In: SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. pp. 1–12. IEEE Press, Piscataway, NJ, USA (2008)
10. Flauzac, O., Krajecki, M., Steffenel, L.: Confiit: a middleware for peer-to-peer computing. Journal of Supercomputing 53(1), 86–102 (July 2010)
11. Flauzac, O., Nolot, F., Rabat, C., Steffenel, L.: Grid of security: a decentralized enforcement of the network security. In: Global, IGI. (ed.) Threats, Countermeasures and Advances in Applied Information Security, pp. 426–443. Manish Gupta, John Walp, and Raj Sharman (Eds.) (April 2012)
12. Keahey, K., Tsugawa, M., Matsunaga, A., Fortes, J.: Sky computing. IEEE Internet Computing 13(5), 43–51 (Sep 2009), `http://dx.doi.org/10.1109/MIC.2009.94`
13. Kelley, I., Taylor, I.: A peer-to-peer architecture for data-intensive cycle sharing. In: Proceedings of the first international workshop on Network-aware data management (NDM '11). pp. 65–72. ACM, New York, NY, USA (2011)
14. Lin, H., Ma, X., Archuleta, J., Feng, W., Gardner, M., Zhang, Z.: Moon: Mapreduce on opportunistic environments. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10). pp. 95–106 (2010)

15. Marozzo, F., Talia, D., Trunfio, P.: A peer-to-peer framework for supporting mapreduce applications in dynamic cloud environments. In: Antonopoulos, N., Gillam, L. (eds.) Cloud Computing, pp. 113–125. Computer Communications and Networks, Springer London (2010)
16. Mastroianni, C., Cozza, P., Talia, D., Kelley, I., Taylor, I.: A scalable super-peer approach for public scientific computation. Future Gener. Comput. Syst. 25(3), 213–223 (2009)
17. Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M.: Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. Computer 45(12), 65–72 (2012)
18. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The eucalyptus open-source cloud-computing system. In: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGrid 2009, Shanghai, China, 18-21 May 2009. pp. 124–131 (2009), `http://doi.ieeecomputersociety.org/10.1109/CCGRID.2009.93`
19. Parashar, M., Pierson, J.M.: Pervasive grids: Challenges and opportunities. In: Li, K., Hsu, C., Yang, L., Dongarra, J., Zima, H. (eds.) Handbook of Research on Scalable Computing Technologies, pp. 14–30. IGI Global (2010)
20. Rottenberg, S., Leriche, S., Lecocq, C., Taconet, C.: Vers une dfinition d'un systme rparti multi-chelle. In: UBIMOB'12 - 8mes Journes Francophones Mobilit et Ubiquit. pp. 178–183 (2012)
21. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). pp. 329–350 (Nov 2001)
22. Satyanarayanan, M.: Mobile computing: the next decade. SIGMOBILE Mobile Computing and Communications Review (15), 210 (Aug 2011)
23. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for vm-based cloudlets in mobile computing. EEE Pervasive Computing (8), 1423 (Dec 2009)
24. Steffenel, L., Flauzac, O., Charao, A.S., Barcelos, P.P., Stein, B., Nesmachnow, S., Pinheiro, M.K., Diaz, D.: PER-MARE: Adaptive deployment of mapreduce over pervasive grids. In: Proc. 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (Oct 2013)
25. Steffenel, L., Flauzac, O., Charao, A., Barcelos, P., Stein, B., Cassales, G., Nesmachnow, S., Rey, J., Cogorno, M., Kirsch-Pinheiro, M., Souveyet, C.: Mapreduce challenges on pervasive grids. J. of Computer Science 10(11) (2014)
26. Steffenel, L.A.: First steps on the development of a P2P middleware for map-reduce (Jun 2013)
27. Vazhkudai, S., Freeh, V., Ma, X., Strickland, J., Tammineedi, N., Scott, S.: FreeLoader: Scavenging desktop storage resources for scientific data. In: Proceedings of Supercomputing 2005 (SC05). Seattle, USA (2005)
28. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.: Improving mapreduce performance in heterogeneous environments. In: Proc. of the 8th USENIX Conf. on Operating Systems Design and Implementation. pp. 29–42. OSDI'08, USENIX Association, Berkeley, CA, USA (2008)