A big data inspired preprocessing scheme for bandwidth use optimization in smart cities applications

Behshad Mohebali[a,*], Amirhessam Tahmassebi[a], Amir H. Gandomi[b], and Anke Meyer-Baese[a]

[a]Department of Scientific Computing, Florida State University, Tallahassee, Florida, USA
[b]School of Business, Stevens Institute of Technology, Hoboken, New Jersey, USA

ABSTRACT

The advancement of Internet of Things (IoT) technologies, such as low-cost embedded single board computers which integrate sensors, communication hardware, and processing power in one unit, has given more traction to the concept of Smart Cities. Having cheaper processing power at their disposal, the sensing units are capable of gathering increasingly larger amounts of raw data locally, which must be processed before being usable. One concern for this scheme is the amount of infrastructure and network bandwidth needed to transfer the data from the acquisition location to a server, which may be miles away, for further processing. The bandwidth available to the sensor network, distributed through the city, is expanding in a lower rate than the size and bandwidth demand of the network it serves. Therefore, transferring the unprocessed data to a central server does not seem feasible unless major compromises are made in terms of data resolution and size. This paper proposes a local big data based preprocessing scheme before the data is transferred to the storage. Using this scheme can free up the network bandwidth, exploit the otherwise wasted local processing power, and release processing load from the central server, allowing it to serve a larger network without the need for more powerful hardware. By making efficient use of network infrastructure the smart city applications are more affordable and scalable.

Keywords: Big Data, Raspberry Pi, Smart Cities, smart grid, Internet of things

1. INTRODUCTION

The emergence of Internet of Things technologies in numerous areas of urban infrastructure such as transportation,[1] energy management,[2,3] agriculture, supply chain, etc., has brought forth opportunities for better situation awareness and more effective decision making on the urban management level based on data.[4] In transportation, the IoT infrastructure can make the cities safer and the traffic flow more fluent using technologies such as RFID plates.[5]

Advancement in several fronts such as:

- **Processors**: The development of low cost, low power, and small processors and system-on-a-chip integrated circuits enables the designers to include processing power in their designs.
- **Communication**: In terms of hardware, the communication modules are taking the same path as the processors, becoming more capable and more affordable. This means that low cost embedded communication systems can be designed around a widely used standard such as Internet Protocol (IP) to integrate as many devices as possible in the same platform.
- **Software**: Having more powerful processors means the smart devices can run more sophisticated operating systems. This will invite in the community of software engineers and developers who are used to develop programs in higher level languages such as Python and Java.

* Corresponding Author: Behshad Mohebali
E-mail: bmohebali@fsu.edu

expands the opportunities in IoT design and development. However, taking full advantage of the power of data requires new approaches to store, transfer, and process data in massive volumes.[6] As the number of smart devices rises, the demand for faster and more efficient communication platforms will become more relevant. The use of the available bandwidth of communication platforms needs to be optimized to ensure that there is no impeding bottlenecks in the design.

Although there is no universally agreed upon definition of the smart cities, the available definitions share the use of information technology resources, such as hardware, software, sensors, and data, to improve the quality of life for residents of an urban area[7] as the fundamental piece. In some sources, the people themselves are counted as a major part of the smart city echosystem.[8]

The idea of exploiting processing power that is geographically distributed into several sites is not new. This concept has been known as **grid computing** in UNICORE project,[9],[10] which was launched in 1999 in an attempt to integrate several German HPC sites and make them accessible to users as one resource as shown in Figure 1. The difference between the UNICORE project and a smart city network of things is that the number of individual sites in UNICORE is much smaller and the processing power in each of them is vastly larger than what we see in IoT nodes. Another significant difference between a grid and a smart city is that the hardware in a computing grid is more likely to be heterogeneous.[11,12]

The rest of the paper is organized as follows. Section 2 describes the methodology that was used in this project. First, the concept of mapReduce which is the most popular paradigm in big data processing is briefly explained. Then, JADE, a framework for developing FIPA (IEEE foundation for Intelligent Physical Agents) compliant agents based on Java, is introduced. This is the framework used for developing the testbed that is explained in Section 3. Section 4 discusses the preliminary results of the testbed performing a simple task of aggregation. At last, the paper is concluded in Section 5.

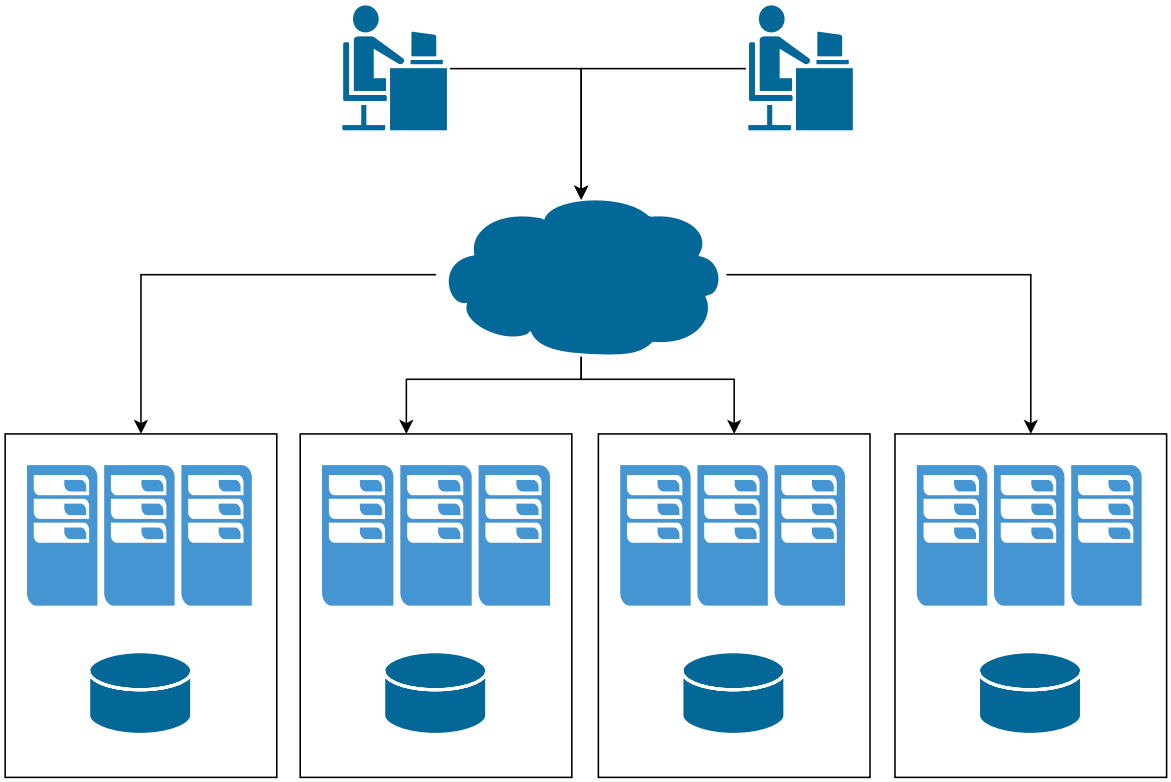

Figure 1. The schematic of a grid computing platform. The aim of the platform is to present geographically distributed processing resources as one resource to the end users.

2. METHODOLOGY

This section explains the approach taken for minimizing the communication necessary between the parties in an IoT network by utilizing the latent processing power in the lower level nodes of the network. In this context, a node is an object with specific dedicated processing power, dedicated sensor (or set of sensors), storage capacity, and ability to make decisions in its domain of influence. Here, the technology of choice for implementing the nodes is JADE (Java Agent Development) framework, which is a Java application programming interface (API). A brief explanation on capabilities of this API is offered in subsection 2.2. Then the topology of the network is explained in subsection 3.

2.1 Map Reduce model

This scheme implements a version of MapReduce programming model[13] that exploits the mostly idle processing power that is local to the sensor data. The MapReduce process is done in five main steps that will execute in sequence. Each step will start after the previous one finishes successfully. However, the tasks within each step may be executed in parallel to improve performance:

1. The MapReduce platform assigns a chunk of the total data to mapper servers and provides them with the data.

2. Each of the mapper servers applies the user defined Map function to the assigned data. The data is in form of a list of pairs as (key1, value1). The output of this phase is another pairs of keys and list of values as (key2, list(value2)).

3. The mappers shuffle the processed output to the reducer servers. The pairs with the same keys coming from different mappers will go to the same reducer.

4. The reducers apply the user defined Reduce() function to the processed data. The input to each call of the Reduce() function is the output of the last step, which is a key and a list of values. The output is a smaller list of values. Mostly just one or zero outputs is expected from each call of the function.[13]

The output of the reduce step will be used to produce the final output of the process. It has to be noted that the domains from which $key1$ and $value1$ are drawn are different than the domain of $key2$ and $value2$.

As an example, consider that the data on the make and model of cars that are passing specific checkpoints is available at all the entrances into the city. Each entrance has a set of embedded sensors that can recognize and record the make and model of all the passing cars. The user wants to know the overall count of the cars entering and exiting the city grouped by their model. In one approach, each embedded sensor will send all its records in batches to the command center for storage and process in specific periods (like every day, or twice a day). This data will be processed using a MapReduce pair of functions at the command center. Another approach is to use the embedded sensors to aggregate their records and send a "summary" of the data, formatted in the way that the command center requires, back to the command center. Each summary will be the result of each embedded sensor applying the MapReduce pair to the batch of data that it has available locally. The summaries need to be reduced one more step at the command center to form the final result. There are several advantages to this approach:

1. The summary sent to the command center is far smaller in size than the data from which it is produced.

2. The processing power of all the embedded sensors are used in the process. In addition, each sensor will process a smaller batch of data.

This is a step in the direction of using all the processing and communication capacity that the network has to offer, shifting the load (both in terms of process and communication) away from the central elements to free them up so that they can handle larger networks.

2.2 JADE framework

JADE[14] is a framework implemented in Java for conveniently and dynamically implement multi-agent systems. It is compliant to the specifications of IEEE Foundation for Intelligent Physical Agents (FIPA), which provides standards to facilitate the interoperability of end-to-end intelligent agents.[15] There are several advantages to using JADE for this case:

- **High level abstraction of functionality**: JADE framework provides the tools necessary for developing agents and establishing communication between the agents by hiding away the implementation of such functionalities from the developer so that the designers and developers can focus on the issues specific to their use case.
- **FIPA compliance**: The FIPA standards are compiled by a body of experts, which gives the developers the prospect of agreeing on a defined set of rules. This means the systems developed using JADE can interact conveniently with any other system that is compliant to the same set of standards.
- **Platform agnostic**: JADE is written in Java and therefore runs on Java Virtual Machine (JVM). This means that the implementation can work seamlessly on any operating system (OS). The agents do not have to run on the same OS or the same hardware. Any system capable of running JVM and connecting to internet can start a JADE platform or subscribe to one regardless of the location of the other members of the platform. This is an important feature for the Internet of Things systems where we have several types of hardware interacting with each other.
- **Open source**: JADE development is an ongoing project that is open to the contributions of the community, which provides continuous modifications and optimization as JADE becomes more popular with Java developers.

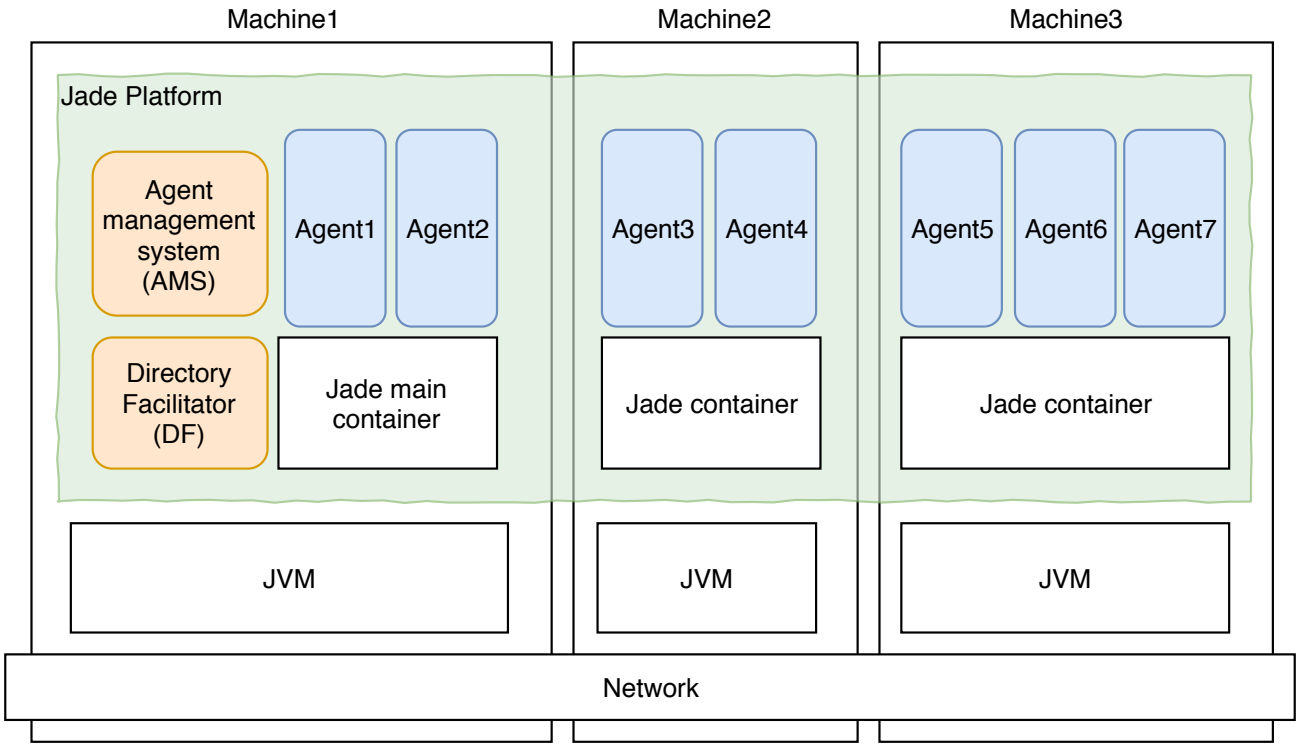

Figure 2. The overall structure of a notional JADE platform that resides on three separate machines and manages seven agents. The communication between the agents is done through the network and is set up by the JADE platform.

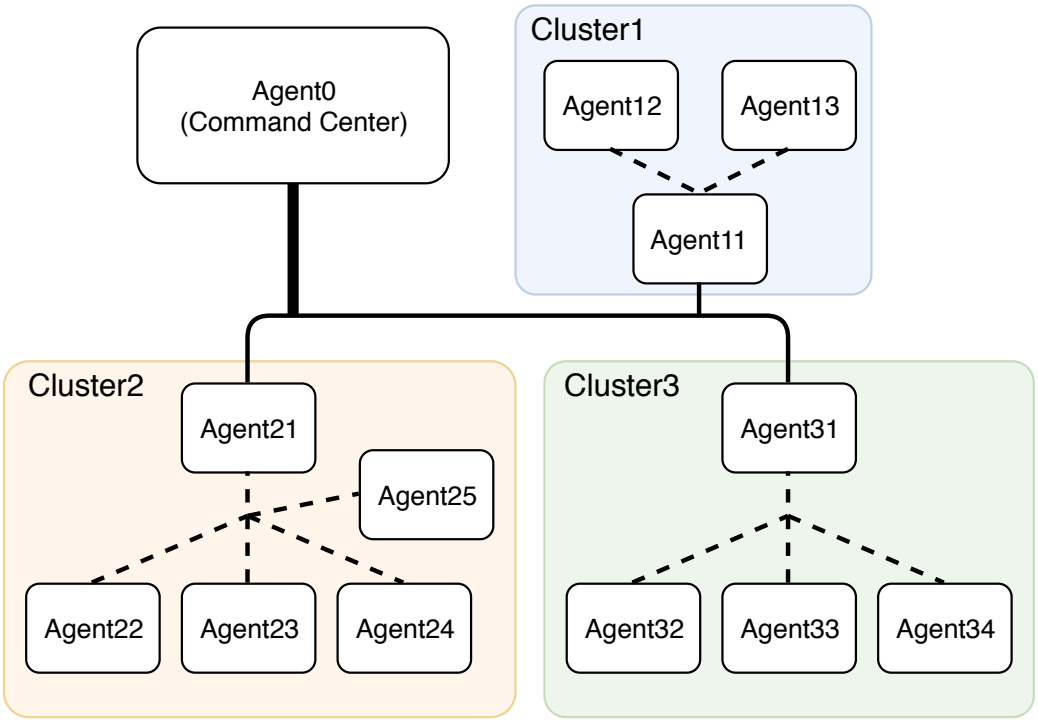

Figure 3. The topology of the system that was used in this project. The platform consists of three groups of agents who communicate together before reporting their aggregated data to the command center, represented here with Agent0.

Figure 2 shows an overview of a notional JADE platform implemented on three machines that are connected to each other through a network. The agents reside within specific JADE objects called "containers." Each JADE platform must have a Main container, an Agent Management System (AMS) object, and a Directory Facilitator (DF) according to FIPA specifications. These two objects are automatically generated by JADE at the startup of the platform. AMS is responsible for providing *white page services*, which involves supervision over the access to the agent platform, bookkeeping on the name and location of the agents. The DF provide *yellow page services*, which involves management of the access and location, and registration of the services within the platform. The agents can register their services (which they are capable of providing to the rest of the platform) with the DF. They can also find agents that can provide specific services using DF yellow page services. User defined agents can be instantiated within the Main container as well. In Figure 2, Agent 1 and Agent 2 are such agents.

Agents which reside on other machines need to be instantiated within a peripheral JADE container that would connect to the JADE platform by subscribing to the main container. Machine 2 and Machine 3 in Figure 2 have their own containers running on their own JVM. The JADE containers, along with the agents inside them, are still considered within the domain on the JADE platform. In this figure, the JVM and the network are depicted outside this domain since they are running independent of the existence of the platform altogether. Any agent, or container (excluding the main container) can be started or destroyed dynamically as the JADE platform runs. However, destroying the main container means the JADE platform is shutting down.

More information about the insides of JADE framework and how to set up an effective platform is available on JADE website[15] and JADE white paper.[14]

3. DESCRIPTION OF THE TESTBED

Figure 3 shows the topology of the testbed. It consists of three groups of agents arranged in clusters with different sizes. Each cluster has a cluster head that takes the requests from teh command center and distributes it to the other agents in the cluster. The agents will process their own gathered data based on the request from the command center and send the result of the process to the cluster head. The cluster head will generate the

collective response of the cluster to the request and sends it to the command center. The command center will stitch the responses from all the clusters and gets the final result of the request that comes from the process done by all the agents in the network.

The cluster head are the agents that generally have more processing power and storage capacity than all the other agents of the cluster. This will give them the ability to redundantly store the data of all the other agents in the cluster. The storage of the cluster data in cluster head will give the cluster one layer of redundancy in case one of the agents becomes unavailable. Since the communication of the agent data with the cluster head happens within the cluster, this will not occupy bandwidth of the command center. The command center agent runs on a PC. All other agents run on individual Raspberry Pi cards with 16 GB storage.

The data used in this case is the 199523 records of Census income from 1994 an 1995.[16] The task is to get the average age of the records in the dataset. The age is the first column of the data and the data is divided between each agent in a random fashion. The number of records per agent is not equal but it is close to 1/12 of the whole number of records. The response of each agent to the request for mean value of the age column is a JSON object. This object contains:

- **Request ID**: so that the command center can distinguish between multiple response coming at the same time
- **Type of the request**: for debugging purposes
- **Name of the agent**: for debugging and documentation purposes
- **Type of the agent**: This property determines whether the response is at the agent level or the cluster level
- **Result object**: This object contains the mean value and the number of records from which the mean value is calculated. The number of records is used at the higher levels to stitch the results together

This information will travel up to the command center so the final result will be the average value of the first column that is calculated by the whole network where each agent has access to one part of the data only.

the cluster heads have two groups of processes. One group for the tasks regarding gathering sensor data and managing data storage. Another group for the tasks related to the management of cluster such as communicating with the agents within the cluster, forming the collective response of the cluster to the command center request, and communicating that response to the command center.

4. RESULTS & DISCUSSION

Table 1 tabulates the preliminary results obtained from the third cluster of the testbed. Each agent, including the cluster head will report the result of the process over their own designated piece of data to the cluster head. Agent31 is the cluster head of the third cluster. That is why there are two rows in the table dedicated to that agent. One for its response as an individual agent, and one for its response as the cluster head (row 5). The responses are in the form of JavaScript Object Notation (JSON).[17] This is the JSON response of the Agent31 as an agent:

Table 1. The results of the cluster 3 agents and the cluster head that aggregates all the individual

agent name	Output	Record Count	Size of data	Size of the response (Characters)
Agent31	34.6025	16270	8.2 MB	152
Agent32	34.3037	15910	8.1 MB	152
Agent33	34.8817	15824	8.0 MB	152
Agent34	34.6380	17590	8.9 MB	152
Cluster3	34.6069	65594	N/A	160
Agent0	34.4942	199523		

```
{
"requestId":1000000012,
"type":"mean",
"agent_type":"node",
"name":"agent31",
"response":{
    "records": 16270,
    "value": 34.6025
    }
}
```


Each agent in the cluster sends an object in this form to the cluster head. This is the JSON response of Agent31 as the cluster head:


```
{
"requestId":1000000012,
"type":"mean",
"agent_type":"clusterHead",
"name":"cluster3",
"response":{
    "records": 65594
    "value": 34.6069
    }
}
```


The last two columns on table 1 show a comparison between the size of the response that is sent to the higher level to the size of the data that is processed.

5. CONCLUSION AND FUTURE WORK

This paper suggests that passing portions of processing burden to the lower level nodes and closer to the data generation location can free up the bandwidth of the command center (where the decisions are made based on the data gathered from the city). The results show a drastic fall in the volume of the data that needs to be communicated with the command center when aggregating the data from a large number of nodes. This reduction can increase the scalability of the network by releasing the limited resource of command center bandwidth, which can be, if not managed properly, a major bottleneck.

Although this approach can reduce the network load on the command center bandwidth, serious questions will remain open. One question is how and with what rate the data is generated. In the case of nodes with regular reading, the rate of data generation is known and manageable. However, if the data generation depends on occurrence of a specific event, the rate can be represented using statistical models. The specifics of such models and the systematic approach for developing them need to be investigated.

One assumption that was made in this paper was that the nodes knew what sort of preprocessing is going to be done on the data. In reality, this will depend on what part of the data is relevant in the grand scheme of things, which is only known by the command center and is communicated to the lower level nodes. In fact, determining what part of the whole processing pipeline can be passed to the lower level nodes (and how to achieve that) has to be decided by the command center.

The other assumption that we made about the network was that the hubs where the results of lower level aggregations are gathered are known based on a prior design decision, which may not be the case in a real network. The choice of hubs based on their physical location, their proximity to the command center, their processing and data storage capacities, and their own bandwidth can affect the overall processing time. More research needs to be done on formulating the problem of hub choice and optimizing the choice for speed, robustness, or data management objectives.

REFERENCES

[1] Sherly, J. and Somasundareswari, D., "Internet of things based smart transportation systems," *International Research Journal of Engineering and Technology* **2**(7), 1207–1210 (2015).

[2] Yan, Y., Qian, Y., Sharif, H., and Tipper, D., "A survey on smart grid communication infrastructures: Motivations, requirements and challenges," *IEEE communications surveys & tutorials* **15**(1), 5–20 (2013).

[3] Bera, S., Misra, S., and Rodrigues, J. J., "Cloud computing applications for smart grid: A survey," *IEEE Transactions on Parallel and Distributed Systems* **26**(5), 1477–1494 (2015).

[4] Cui, X., "The internet of things," in [*Ethical Ripples of Creativity and Innovation*], 61–68, Springer (2016).

[5] and T. A. Rahman and Rahim, S. K. A., "Rfid vehicle plate number (e-plate) for tracking and management system," in [*2013 International Conference on Parallel and Distributed Systems*], 611–616 (Dec 2013).

[6] Marjani, M., Nasaruddin, F., Gani, A., Karim, A., Hashem, I. A. T., Siddiqa, A., and Yaqoob, I., "Big iot data analytics: architecture, opportunities, and open research challenges," *IEEE Access* **5**, 5247–5261 (2017).

[7] Al Nuaimi, E., Al Neyadi, H., Mohamed, N., and Al-Jaroodi, J., "Applications of big data to smart cities," *Journal of Internet Services and Applications* **6**(1), 25 (2015).

[8] Galán-García, J. L., Aguilera-Venegas, G., and Rodríguez-Cielos, P., "An accelerated-time simulation for traffic flow in a smart city," *Journal of Computational and Applied Mathematics* **270**, 557–563 (2014).

[9] Erwin, D. W. and Snelling, D. F., "Unicore: A grid computing environment," in [*European Conference on Parallel Processing*], 825–834, Springer (2001).

[10] Romberg, M., "The unicore architecture: Seamless access to distributed resources," in [*Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No. 99TH8469)*], 287–293, IEEE (1999).

[11] Tahmassebi, A., "ideeple: Deep learning in a flash," in [*Disruptive Technologies in Information Sciences*], **10652**, 106520S, International Society for Optics and Photonics (2018).

[12] Mohebali, B., Tahmassebi, A., Gandomi, A. H., Meyer-Baese, A., and Foo, S. Y., "A scalable communication abstraction framework for internet of things applications using raspberry pi," in [*Disruptive Technologies in Information Sciences*], **10652**, 1065205, International Society for Optics and Photonics (2018).

[13] Dean, J. and Ghemawat, S., "Mapreduce: simplified data processing on large clusters," *Communications of the ACM* **51**(1), 107–113 (2008).

[14] Bellifemine, F., "Jade-a white paper," *exp* **3**(3) (2003).

[15] "Jade tutorials and guides." http://jade.tilab.com/documentation/tutorials-guides/. Accessed: 2019-03-01.

[16] Dua, D. and Graff, C., "UCI machine learning repository," (2017).

[17] "Json (javascript object notation)." https://www.json.org/. Accessed: 2019-03-01.