

SkyData: Rise of the Data How Can the Intelligent and Autonomous Data Paradigm Become Real?

Etienne Mauffret^{id}^a, Elise Jeanneau^{id}^b and Eddy Caron^{id}^c

UMR CNRS, ENS Lyon, UCB Lyon 1, Inria 5668, Lyon, France

fi

Keywords: Distributed Systems, Autonomous Data, Data Management, Multi-Agents System.

Abstract: With the rise of Data as a Service, companies understood that whoever controls the data has the power. The past few years have exposed some of the weaknesses of traditional data management systems. For example, application owner can collect and use data to their own advantage without the user's consent. We introduce in this paper the SkyData concept, which revolves around autonomous data evolving in a distributed system. This new paradigm is a complete break from traditional data management systems. In this paper we will show a way to define autonomous data as well as some challenges associated with their specificities.

1 INTRODUCTION


A fundamental characteristic of our era is the deluge of *Data*. Data is everywhere, *e.g.*, scientific, entertainment and IoT applications, personal activity trackers, *etc.* Since Grid environments, data management in distributed environments is quite a common practice (Filangi et al., 2008). Nowadays, Clouds provide many solutions to store data (Strauch et al., 2011). A data manager can be defined through its functionalities which can be understood as *services*. Indeed, many services are required for data management, such as security (Kaufman, 2009), replication strategy (Lei et al., 2008), data migration (Kang and Reddy, 2008), green data transfer (Orgerie and Lefèvre, 2013), synchronization (Filangi et al., 2008). Many ways to design those services exist, and each data manager includes its own point of view to compose these services (Strauch et al., 2011). Thus, when users need a specific data management strategy, they can choose an appropriate data manager but with high dependency to this manager.


Data management since its genesis uses a central-


ized data manager as a standard (Han et al., 2011). With huge and distributed system, new solutions were designed for Grid (Sakr et al., 2011) or P2P system (Antoniou et al., 2006) for example. More recently, researchers have proposed solutions to deal with dynamic use, for example, tools that provision data management system on top of storage devices (Tessier et al., 2020). Usually, the point of view of data management is centered on applications rather than data, even when an *automatic* solution is provided. For example, the authors of (Kumar et al., 2014) deal with data placement for intensive web applications.

Nonetheless, many approaches are relevant and efficient but domain specific and provide full control of actual data to the data management system built by the application. Despite unquestionable benefits to users, Cloud computing raises several concerns about data management. Companies have understood that whoever controls the data has the power and control over the business. For example, the company that built the data management system of an application can use it to sell data or to collect personal data or use any (or every) data to their own advantage, without needing any permissions or consent from users.

SKYDATA aims to prevent those abuses and proposes a new take on data management: the *self-managed* data. Each replica of each piece of data is carried by an autonomous agent that acts on its own on the behalf of the application. The association of the agent and the replica is called SkyData, or SKD. As autonomous entities, SKD are capable to make decision to manage their own data, thus SKD are self-

^a  <https://orcid.org/0000-0001-8444-6293>

^b  <https://orcid.org/0000-0003-1924-2280>

^c  <https://orcid.org/0000-0001-6626-3071>

This research was funded, in whole or in part, by l'Agence Nationale de la Recherche (ANR), project ANR-22-CE25-0008-01. For the purpose of open access, the author has applied a CC-BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

managed data. With this paradigm shift, each self-managed data is controlled by none but itself in a distributed manner, independently of any data management service. One of the important aspect of such system is the non existence of a global index table that could indicate the name (and position) or every data in the system. Rather, a SkyData environment can only have some partial index table that hold a limited amount of reference of SKD. Any reference to a SKD is based on a registration initiated by the SKD and can be cancelled at any time.

Hence, this paradigm does not aim to replace existing data management systems, it should be viewed as an alternative and a complete rupture from them and should establish new connections between data management, distributed systems and multi-agent systems. This paper present the core concepts and structures of self-managed data and their environment. However, we ask the readers to keep in mind that this is a new concept where data can be assimilated to a living entity without control. Therefore, we present here a generic definition of the model needed, which may need to be adapted for specific use cases. Self-managed data are able to decide, from their own knowledge about the system, which action to undertake: they could decide to replicate, move to another physical node, leave the system, *etc.* Like mobile agents, self-managed data can move from one host to another and continue its execution on the new host. In the context of network management, it has been shown that mobile agents provide more robust and fault-tolerant solutions while allowing reduction of the network load (Lange, 1998). In the context of data management, the number and position of each replica of data has proven to be a key factor for good performances (Kaur et al., 2019) while being a hard problem (Golubchik et al., 2009) Self-managed data must then be able to consider both fields in order to provide good services.

In this paper, we present SKYDATA, the first proposal to design an environment for self-managed data. We present the state of the art and the context in which this paradigm came to existence (Section 2), then we define the SKYDATA Environment structure and environment in (Section 3). We then discuss of capabilities and goals of SKD (Section 4). We then conclude and discuss the SKYDATA paradigm (Section 5).

2 RELATED WORK

We can define four major eras for the data management in IT (Information Technology). The early era was the *digitalization of information* (Sendov,

1997). The second era was the *building of Database Management Systems* (DBMS). While many of them were based on SQL, nowadays, DBMS continue to grow with the amount of data, and solutions based on NoSQL approaches have appeared (Leavitt, 2010; Moniruzzaman and Hossain, 2013).

Next, the emergence of distributed management systems (Moysiadis et al., 2018) has arisen and evolved through Grid, P2P and Cloud architectures: the third era around *distributed data management* was born. Technical comparison of NoSQL data management systems can be found in (Bunch et al., 2010; Han et al., 2011). Those studies compare the different approach chosen for data management systems, as well as a study on their performances.

The fourth era is the *data storage* era. The ascent of DaaS (Data as a Service) systems (Kravchenko et al., 2019) is undeniable, and its success can be observed through many solutions like Dropbox, Google Drive, S3 from Amazon, iCloud from Apple, OneDrive from Microsoft, Owncloud, *etc.* Data is now everywhere and is therefore at the heart of our IT life. Emerging from the foundations of these four era is the advent of a today well-known era that we could call the *data control era*.

Through SKYDATA, we have the ambition to contribute to the genesis of a fifth and new era of data management: the *self-managed data era*. All knowledge around data management such as data modeling, storage, migration, and encryption will be useful. Data control will be delegated to data themselves. Thus, we plan to design a solution without middleware nor data manager. For those reasons, existing data management systems seem really far from what we propose in this project and comparison hardly relevant. Nevertheless, not so far from our main ideas, many researchers have introduced data-centric solutions (Polese et al., 2020; Provost and Fawcett, 2013) and we have initiated new research on data-driven service discovery in (Houmani et al., 2020). Other researchers have focused on using autonomous data (Caragea et al., 2003), but those solutions have kept a certain degree of control over the data.

The SKYDATA stated goal of providing data storage independently of any data management service warrants comparison with blockchain technology, which is used to maintain a distributed ledger independently of any centralized authority. While blockchain is indeed capable of maintaining the consistency of a small amount of data in a truly decentralized manner, the amount of data that can be stored in a block is very limited. This is notably the reason Non-Fungible Tokens (NFTs) are used to store URIs to a resource, rather than storing the resource itself.

Additionally, blockchain scalability is still an open problem (Zhou et al., 2020), which prevents it from acting as large scale distributed data storage. While blockchain technology is not itself a viable alternative to SKYDATA Environment, it might be a useful tool to implement some functionalities of the SKYDATA Environment Project, notably for the certification and traceability of self-managed data.

The foundations of self-managed data are inspired from several existing paradigms, including multi-agent systems (Russell, 2010), P-systems (Paun, 1999) and population protocols (Angluin et al., 2004). However, the spirit of SKYDATA is quite different from that of multi-agent systems. In multi-agent systems, autonomous entities take individual decisions in order to achieve a common objective or self-interested objectives. Deliberative agent are able to make complex decisions by reasoning on an explicit representation of their environment. In large scale and dynamic environments where the agents usually have partial observability of the system, building this model of the environment and making coordinated decisions is very difficult (Beynier and Mouaddib, 2014). In the autonomous data paradigm, a SKD aims at using the collective intelligence to achieve its own selfish goals, for example its survival in the system. The other difference lies in the fact that both data and code are embedded. The latter opens new ways of computing and composing services, which are rather close to the P-system paradigm. Finally, the distributed autonomous load-balancing between SkyData and their replicates will be based on probe mechanisms that are close to the counting problem in population protocols.

3 THE SkyData ENVIRONMENT

SKYDATA Environments are the first model of the new paradigm of self-managed data. Hence it comes with many news concepts and definitions. In this section, we first present the specific definitions related to SKYDATA Environments before using those definition to introduce the core concepts of SKYDATA Environments.

3.1 Short Definitions

- SKD: SkyData, noted SKD, are autonomous agents that carry the data itself and metadata. Among the metadata, some code allows the SKD to plan for its actions. SKDs are endowed with goals and capabilities making it autonomous, proactive and reactive within the system.
- Family (of SKDs): As in many distributed system,

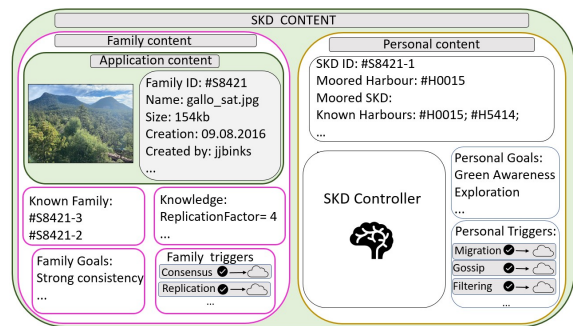


Figure 1: Example of content of a SKD.

stored data is replicated. When a SKD is replicated, a new SKD containing a replica is created. The set of SKDs that hold the same piece of data is called a family.

- SKW: SkyWorkers, noted SKW, are another entity of a SKYDATA Environment. They can be viewed as special SKDs that are entrusted with a system task rather than application data. They are capable of moving within the system and communicating with SKDs to accomplish their task.
- Harbour: Harbours are the physical host that hold SKDs and SKWs. They act as an interface for users. In order to keep SKDs as autonomous as possible, harbour s have a very limited amount of capabilities.
- Moored (SKD): A SKD that is hosted by a harbour is said to be moored to that harbour . SKDs hosted by the same harbour are said to be moored together.
- SKYDATA Environment: A SKYDATA Environment is a collection of SKDs, SKWs and harbour s. SKDs and SKWs can interact by sending each other messages, and can migrate from a harbour to another.

In addition to those definitions, we define some core mechanism of SKYDATA Environments.

- Triggers: SKDs have a collection of triggers that are used to check associated boolean conditions. If the condition is verified, the SKD then reaches the corresponding micro-service node and executes a remote procedure call. This mechanism is used to implement the capabilities of SKD.
- Internal Controller: Each SKD has an internal controller that represents its intelligence. Among others tasks, The controller uses goals to determines which triggers to check.

3.2 The SKD Concept

SKDs are agents that carry a piece of data and are

thus the core entities of the paradigm. They are autonomous, proactive, reactive and have some social ability, as defined in (Wooldridge and Jennings, 1995). Each SKD is endowed with goals and capabilities that will influence its planning and the action that it will undertake. Its behavior is influenced by the (partial) knowledge of the environment and a change of this environment can trigger a response from the SKD. The goals of a SKD are set at its creation and usually aim to provide a good quality of service. However, SKDs are capable of learning and their goals can evolve during the run to meet new applicative needs. As a result, SKDs can be seen as application-independent, since they might participate in different applications over time. Intuitively, a SKD should not be seen as a worker made to accomplish a task, but rather as a “person” that might work different jobs during their lifetime.

We can also use the Beliefs-Desires-Intentions model, presented in (Bratman et al., 1988) to describe the self-managed data. Each SKD has its own beliefs, its own partial view of the environment and will choose amongst its desires, or goals, to undertake actions among the set of its capabilities.

An SKD is identified by its family ID (which is common to the family, but otherwise unique within the system), and its SKD ID (which is unique within the family).

An SKD can register itself to some services as well, such as a logical group to accomplish some common objectives or a partial index table for example. Such table allows a user to keep track of data it owned. SKD remains in control of their registration and can leave any group or service at any time.

Family Content: SKDs that belong to the same family share a part of their metadata. This section of the metadata is referred to as the family content. It contains, among other information, all the information needed by users : the data itself, knowledge about the data (such as the name of the file, its size, *etc.*), and the family ID. The family content also includes internal information (inaccessible by users) such as a list of members of the family and some knowledge about the data or the system. Some goals, such as the consistency guarantees, must be shared by the family in order to satisfy the applications guarantees. Finally, some triggers and the associated micro-services are grouped actions, such as a consensus protocol. Those triggers and associated services must then be compatible within a family. The family content is shared in a distributed way and will converge through time to provide properties of the family.

Personal Content: Additionally to the family content, SKD has a personal content. Unlike the fam-

ily content, the personal content is not shared and can be different from a SKD to another, even within the same family. This is where the individuality of the SKD is expressed. The personal content is composed of the internal controller and some personal goals, triggers and knowledge. For example, the migration trigger (or associated micro services) does not need to be shared with the family and is included in the personal content: this means that each SKD in the family may have a different placement strategy.

An example of SKD is illustrated in the Figure 1. In this example, the data itself is a satellite picture of a mountain of size 154kb created by jjbinks on the 09/08/2016. Its family ID and SKD ID are #S8421 and #S8421 – 1 respectively. It is currently in the harbour #H0015 and knows some other harbour s as well as 2 other member of its family (#S8421 – 3 and #S8421 – 2). The example also shows some family goals and family triggers as well as personal ones.

3.3 The SKW Concept

SKWs are entities created to accomplish one precise task for the system, such as gathering some information or aggregate learning models. SKWs can be viewed as a dataless variation of SKDs: they do not hold any data but otherwise use the same mechanisms. However, their life cycle is not quite the same, as SKW are created to accomplish one precise task and disappear once it is done. SKWs largely share the same structure as SKDs: their family content is composed by knowledge, triggers and goals as well as the task to be accomplished. They also have personal content with an internal controller. Hence, despite the task to accomplish, SKWs are autonomous and will choose freely how to best accomplish their task.

Some operations require more knowledge or capabilities that an SKD has on hand. In this case, the SKD can invoke a family of SKWs to accomplish a task to help it perform this operation. For example, some learning services require to gather models from as many SKDs as possible to aggregate them and provide a trained model. To be able to gather those models without disturbing its other goals, a SKD can create a family of SKW that will explore the system to collect local models and provide a trained model.

It should be noted that in most situation, SKWs do not have authority over the SKDs: they can talk to SKDs and carry information between them, but the SKDs are not forced to cooperate with the SKWs. This enables us to preserve the autonomy of the SKDs while still making use of task-centric agents. However, in rare cases and with the permission of a system administrator, it is possible to create some SKWs

with the capability to modify or destroy SKDs. This kind of SKW can be seen as “super users” if the system needs some administrative control, for example to delete some illegal data.

An example of SKW is illustrated in Figure 2. The SKD, identified by #W0333 – 2 has some knowledge shared among the family as well as personal content and an internal controller.

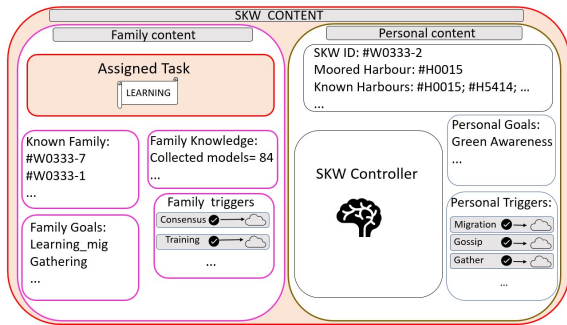


Figure 2: Example of content of a SKW.

4 CAPABILITIES

We describe in this section some fundamental operations and capabilities that any SKD should be able to perform. This section is not exhaustive and aims to help grasp the concept and possibilities brought by a SKYDATA Environment. Because SKWs have similar capabilities, everything that is discussed in this section about SKDs also applies to SKWs.

4.1 Communications

A SKYDATA Environment is an unknown dynamic distributed system: SKD can join or leave the system at any time for numerous reasons. Upon joining the system, a SKD does not have any knowledge about the system and must use sensors and communication to gather knowledge. We consider that every SKD have a protocol to reach for the local harbour and moored s SKD. As harbours can provide the list of local SKD, it is possible for two SKDs to communicate with each other as long as they remain in the same harbour. Therefore, each harbour constitutes a clique of SKDs that can all communicate with each other. We refer to such a group as a *moored clique*. This clique has dynamic membership, since any of its members may at any time start a migration or create a replica of itself.

Moreover, SKDs keep tracks of their family members. Indeed, as members of a family share their family content (the data itself, some goals, etc.), they will be required to communicate with each other in order

for most applications to behave normally, for example to establish a consensus to maintain strong consistency among the replica. As a result each family also constitutes a clique, referred to as the *family clique*, as every member of a family can contact every other member of the family. Due to replication and failures, this clique also has dynamic membership. We consider that a family must maintain the family clique as much as possible and describe in the respective sections the step needed to do so. A SKYDATA Environment can be highly dynamic and can include many SKDs at any given point in time. As SKDs can migrate from a harbour to another, it should carry as little information as possible. Therefore, it is both difficult and undesired for a SKD to store information about every other SKD in the system. A SKD can only communicate (i.e., send a message) with another SKD if it knows its location, and thus with the two cliques previously described. Those two cliques can be used to implement distributed algorithms, although most algorithms would need to be adapted to the specificity of the system and will be the subject of future works. Figure 3 provides a visual representation of a moored clique and a family clique, with 6 SKDs within 3 harbours. The family #S076 counts 3 replica, while others are unique in the system. This example highlights the two cliques that SKD #S076 – 4 belongs to.

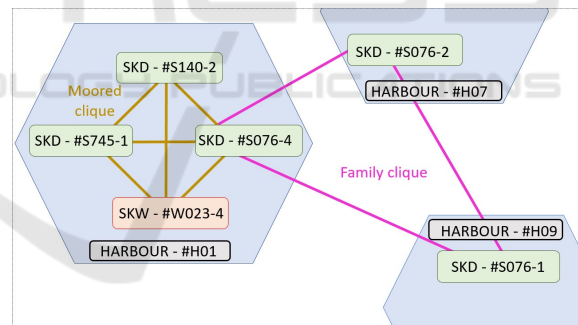


Figure 3: Example of a SKYDATA Environment with a SKD point of view.

It is important to note and understand that those limited communications do *not* allow for a SKD or a harbour to reach a given SKD deterministically. Indeed, due to the high dynamics of the system, the autonomy of the SKD and limited communications, it is not possible to guarantee that a path exists between two SKDs.

This particularity causes a fundamental difference with traditional data management systems. Indeed, it is impossible to ensure that a user will be able to obtain some specific data. Therefore, we insist that SKYDATA Environments does not aim to replace traditional data management systems, but aim to offer an

alternative when it is suited to the application.

4.2 Replication and Deletion

As part of a data management system, a SKD aims to maintain the replication factor of the data it holds. In others words, the SKD tries to maintain the family size to some value. The good news is that the replication factor should be a family goal and thus every member of the family aims for the same family size. However, it could occur that some SKD is wrong about the size of its family, for example in case of a family partition. In this situation, the SKD that holds this incorrect belief might either replicate or delete itself to help the family to reach the desired goal. In order to prevent too many replications or deletions, we add some controls to those protocols. We use the Belief-Desire-Intent model to describes the plan to maintain the replication factor (i.e., the family size).

We assume that every SKD has some beliefs about the replication factor and the current family size, noted RF and N respectively, and can communicate with known members of its family. Upon update of one of those values, if the SKD believes that $RF \neq N$, it will intent to execute the replication or deletion protocols, described in Algorithm 2 according to the plan described in Algorithm 1. The plan states that if there are too few member in the family (i.e., $N < RF$) then the SKD must run the replicationProtocol, on the other hand, if there are too many family member, the deleteProtocol is initiated. The replication protocol determines if the SKD will replicate with a given probability p_1 ¹ and will send a message with the ID of the new replica once created.

In order to delete itself, a SKD asks for permission to every known member of its family and waits for N confirmations. Upon reception of a deletion permission request, SKD can send one answer among: *ack*, *denial* and *undecided*. An *ack* is sent if the SKD that responds to the request does not also want to delete itself, and if it believes there are enough family members. The SKD sends *undecided* if it wants to delete itself. Finally, a *denial* is sent if there are not enough family members or with some probability p_2 ¹ instead of an *undecided*. Upon reception of a *denial*, the SKD forfeits its wish to delete itself and thus can send a confirmation to other SKDs, which helps prevent deadlocks. Once a SKD receives a sufficient number of *acks*, it simply deletes itself. All of the previous steps are then repeated until every SKD that wanted

¹We recommend $p_1 = \frac{RF-N}{N}$ and $p_2 = 1 - \exp(\frac{\ln(1-\frac{RF}{N})}{N})$ for the appropriate expected number of replications and deletions.

to delete itself has either succeeded or given up. This protocol ensures that at least RF SKDs will not delete themselves, while avoiding deadlocks in most cases. Moreover, this protocol does not require every member of the family to have the same belief about the size of the family.

The function *replicate()* creates an exact copy of the SKD as a new SKD and returns the SKDID of the new copy. The *delete()* call simply erases the SKD from the system. Algorithms 1 and 2 assume that the family uses safe migrations to prevent family partitions.

Algorithm 1: Maintain Replication Factor Plan.

Beliefs: replicationFactor(RF)
familySize(N)

```

+!maintainRF : N < RF
  1 ← .replicationProtocol(RF, N)
+!maintainRF : RF < N
  2 ← .deleteProtocol(RF, N)

```

Algorithm 2: REPLICATION Protocol.

Code for a SKD with $SKDID = i$.

```

void replicationProtocol (RF, N)
  1 | do with proba  $p_1$ :
  2 |    $ID_{Replica} \leftarrow replicate()$ 
  3 |   Broadcast(new,  $ID_{Replica}$ ) to family
void deleteProtocol (RF, N)
  4 |  $deleteMyself \leftarrow True$ 
  5 | while(True):
  6 |   Broadcast(delete, i) to family
  7 |   wait for N message:
  8 |   if (at least RF ack): delete()
  9 |   if (at least 1 denial):
 10 |      $deleteMyself = False$ 
 11 |     break
on reception of (delete, j)
 12 | if (! $deleteMyself$ ) : send(ack) to j
 13 | elif ( $RF < N$ ) : send(denial) to j
 14 | elif with proba  $p_2$  : send(denial) to j
 15 | else: send(undecided) to j

```

4.3 Gathering and Broadcasting Information

An important capability required for numerous algorithms, notably the learning process, is the capability to gather and broadcast information around the system. Indeed, a SKD can only have a partial view of the system at any time. And having a better representation

of the system may allow for better decisions. We propose two different approaches to perform a gathering process, each with pros and cons.

The first approach is the gossip based algorithm. In this protocol, each SKD periodically sends its knowledge among the two cliques it belongs to (moored clique and family clique). Symmetrically, it will receive an update from every member of its family and every SKD in its harbour. These steps are then repeated over and over, thus spreading information through gossip. The advantage of this approach is that each SKD will quickly have most of the needed knowledge to be able to take good decisions. However, the number of messages scales badly with the number of SKDs and harbours and much information is shared redundantly. It is possible to perform the same algorithm but for only a handful of SKD at the cost of a loss of knowledge for every other SKD.

The second approach relies on SKWs. A SKW (or a family of SKWs) specialized in gathering and sharing knowledge can migrate from harbour to harbour with the task of communicating with as many SKDs as possible. When the SKW arrives in a harbour, it shares its knowledge of the system and requests knowledge from local SKDs. This protocol allows gathering and propagate knowledge while keeping a low communication cost. However, a given SKD will only receive information upon meeting the SKW. This process can take a long time and could lead to the incapacity for a SKD to receive information, in the event it does not meet any SKW. This issue can be mitigated by combining this technique with the first algorithm, for example by spreading some limited gossip when the SKW arrives in a new harbour.

Those two approaches are early attempts to establish gathering and broadcasting capabilities in a SKYDATA Environment. Future works will address this challenge and propose a more efficient way to gather and broadcast knowledge through gossip and/or SKWs.

5 DISCUSSION & CONCLUSION

In this paper, we presented SKYDATA, a first attempt at self-managed data, a new paradigm that distances itself from traditional data management systems. This paradigm is born from the many issues associated with traditional data management systems, such as resells or private information collected without consent, for example. Self managed data, or SKDs, are agents endowed with data, capabilities and goals to achieve. They are free to behave as they wish and try to accomplish their goals as efficiently as possible. They use

learning algorithms to improve their decision making and learn new capabilities and services. We discussed how SKDs could be developed and provided some insight on useful capabilities.

SKDs are independent and autonomous entities, and therefore cannot be used as a traditional data management system. Additionally, the size and dynamics of a SKYDATA Environment makes it difficult for SKDs to keep a complete view of the system. As a result, it is impossible to centralize up to date and complete knowledge of the system. In particular, it is impossible for a SKD to reach another remote SKD if they are not from the same family. This very fact creates a break from existing data management systems, making the classic “join request” impossible to achieve deterministically. Moreover, the capacity of SKDs to replicate, leave the system or migrate make them different than traditional multi agents systems.

A first version of a SKYDATA Environment is being developed using JADE and JASON to implement SKDs using the Beliefs-Desires-Intentions model with AgentSpeak (Bordini et al., 2007). This prototype will be presented in future works.

Many challenges rise from the lack of deterministic point-to-point communications, except for limited groups such as the family clique and the moored clique, and the lack of a complete view of the system. For example, we do not yet know how to implement a deterministic broadcast or gathering. Many distributed algorithms and services must be adapted for this new paradigm. This difficulty also makes many existing applications unadapted to a SKYDATA Environment. We recall that autonomous data do not aim at replacing traditional data management systems in every situation, but seems to be a desirable alternative whenever it is possible.

REFERENCES

- Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M. J., and Peralta, R. (2004). Computation in networks of passively mobile finite-state sensors. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*.
- Antoniou, G., Bertier, M., Bougé, L., Caron, E., Desprez, F., Jan, M., Monnet, S., and Sens, P. (2006). GDS: An architecture proposal for a grid data-sharing service. In *Future Generation Grids*. Springer.
- Beynier, A. and Mouaddib, A.-I. (2014). Applications of dec-mdps in multi-robot systems. In *Robotics: Concepts, Methodologies, Tools, and Applications*.
- Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons.

- Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational intelligence*.
- Bunch, C., Chohan, N., Krintz, C., Chohan, J., Kupferman, J., Lakhina, P., Li, Y., and Nomura, Y. (2010). An evaluation of distributed datastores using the appscale cloud platform. In *2010 IEEE 3rd International Conference on Cloud Computing*. IEEE.
- Caragea, D., Silvescu, A., and Honavar, V. (2003). Decision tree induction from distributed heterogeneous autonomous data sources. In *Intelligent systems design and applications*. Springer.
- Filangi, O., Beausse, Y., Assi, A., Legrand, L., Larré, J.-M., Martin, V., Collin, O., Caron, C., Leroy, H., and Allouche, D. (2008). Biomaj: a flexible framework for databanks synchronization and processing. *Bioinformatics*.
- Golubchik, L., Khanna, S., Khuller, S., Thurimella, R., and Zhu, A. (2009). Approximation algorithms for data placement on parallel disks. *ACM Transactions on Algorithms (TALG)*.
- Han, J., Haihong, E., Le, G., and Du, J. (2011). Survey on nosql database. *2011 6th international conference on pervasive computing and applications*.
- Houmani, Z., Balouek-Thomert, D., Caron, E., and Parashar, M. (2020). Enhancing microservices architectures using data-driven service discovery and qos guarantees. In *2020 20th IEEE/ACM CCGRID*. IEEE.
- Kang, S. and Reddy, A. N. (2008). User-centric data migration in networked storage systems. *2008 IEEE ISDP*.
- Kaufman, L. M. (2009). Data security in the world of cloud computing. *IEEE Security & Privacy*.
- Kaur, A., Gupta, P., Singh, M., and Nayyar, A. (2019). Data placement in era of cloud computing: a survey, taxonomy and open research issues. *Scalable Computing: Practice and Experience*.
- Kravchenko, Y., Leshchenko, O., Dakhno, N., Trush, O., and Makhovych, O. (2019). Evaluating the effectiveness of cloud services. In *2019 IEEE ATIT*. IEEE.
- Kumar, K. A., Quamar, A., Deshpande, A., and Khuller, S. (2014). Sword: workload-aware data placement and replica selection for cloud data management systems. *The VLDB Journal*.
- Lange, D. B. (1998). Mobile objects and mobile agents: The future of distributed computing? In *European conference on object-oriented programming*. Springer.
- Leavitt, N. (2010). Will nosql databases live up to their promise? *Computer*.
- Lei, M., Vrbsky, S. V., and Hong, X. (2008). An online replication strategy to increase availability in data grids. *Future Generation Computer Systems*.
- Moniruzzaman, A. and Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*.
- Moysiadis, V., Sarigiannidis, P., and Moscholios, I. (2018). Towards distributed data management in fog computing. *Wireless Communications Mobile Computing*.
- Orgerie, A.-C. and Lefèvre, L. (2013). Energy-efficient data transfers in large-scale distributed systems. In *Handbook of Energy-Aware and Green Computing, Volume 2*. Chapman and Hall/CRC.
- Paun, G. (1999). P systems with active membranes: Attacking np complete problems. Technical report, Department of Computer Science, The University of Auckland, New Zealand.
- Polese, M., Jana, R., Kounev, V., Zhang, K., Deb, S., and Zorzi, M. (2020). Machine learning at the edge: A data-driven architecture with applications to 5g cellular networks. *IEEE TMC*.
- Provost, F. and Fawcett, T. (2013). Data science and its relationship to big data and data-driven decision making. *Big data*.
- Russell, S. J. (2010). *Artificial intelligence a modern approach*. Pearson Education, Inc.
- Sakr, S., Liu, A., Batista, D. M., and Alomari, M. (2011). A survey of large scale data management approaches in cloud environments. *IEEE communications surveys & tutorials*.
- Sendov, B. (1997). Towards global wisdom in the era of digitalization and communication. *Prospects*.
- Strauch, S., Kopp, O., Leymann, F., and Unger, T. (2011). A taxonomy for cloud data hosting solutions. *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*.
- Tessier, F., Martinasso, M., Chesi, M., Klein, M., and Gila, M. (2020). Dynamic provisioning of storage resources: a case study with burst buffers. In *2020 IEEE IPDPSW*. IEEE.
- Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *The knowledge engineering review*.
- Zhou, Q., Huang, H., Zheng, Z., and Bian, J. (2020). Solutions to scalability of blockchain: A survey. *IEEE Access*.