Scalable Computing: Practice and Experience
Volume 12, Number 3, pp. 351–362. http://www.scpe.org
ISSN 1895-1767

AN ADAPTIVE AND SCALABLE REPLICATION PROTOCOL ON POWER SMART GRIDS*

JOAN NAVARRO† JOSÉ ENRIQUE ARMENDÁRIZ-IÑIGO‡ AND AUGUST CLIMENT §

Abstract. Cloud based storage systems are known to provide high scalability and reliability overcoming the traditional constraints of static distributed systems. The processing capacity over thousands of machines makes this approach especially suitable for many environments. In particular, we focus on power networks. These systems are currently decentralizing their architectures due to the growth of renewable sources and the increasing power demand which are obstacles to the traditional power network radial distribution. This new decentralized architecture, which demands computing abilities for network monitoring and improving customer services, is denoted as power smart grid. This paper proposes a new scalable dynamic storage architecture and its associated replication protocol, with its correctness proof, aimed to store data with different consistency levels. In addition it is able to perform some parallel data computations adapting itself to the physical and dynamic power smart grid layout.

Key words: Dynamic systems, cloud computing, eventual consistency, smart grids, replication.

1. Introduction. Power networks are demanded to be highly reliable and available because they have to supply all the infrastructures of a country at anytime and anywhere. This prevents power companies from updating and improving their systems because most of the changes may seriously affect critical services they are currently providing since novel devices might not be as tested as older ones. This leads to inefficient—due to their centralized nature—schemes which are expensive and even harder to maintain and scale.

With the growth of renewable energies the power network centralized model is not only able to scale but also cannot work properly; the aforementioned renewable energy sources behave different than traditional sources. Moreover, current power networks are not able to remotely monitor power consumptions on the low voltage (LV) network which prevents companies from building new business strategies fitted to the end user needs [8]. This situation urges a substantial change which consists of decentralizing the power network and building a distributed system able to fulfill the current society requirements and technologies.

Recently, this new paradigm has also been referred to as power smart grid (intelligent grid). The goal of a smart grid is to take advantage of the current digital technologies and build up an intelligent information system over all devices within the power network: from suppliers to consumers. This might allow companies to efficiently tune the power distribution and route energy where and when it is needed.

Smart grids have been a hot topic during the last few years and several approaches have been proposed: the gridSmart project [1] proposes an upgrade of the Ohio electric grid by using digital communications and automated functioning. This permits customers showing how smart grid technologies provide customers with greater energy control. It can also improve electricity delivery and cut energy consumption to delay the need to build more power plants. The Masdar Eco city [23] project proposes to build a energetically sustainable city in Abu Dhabi. IBM and Malta's government are powering a project [24] which consists in transforming the distribution network to improve operational efficiency and customer service levels by changing the current electricity meters to smart devices and connecting them to an information system enabling remote reading, management, and monitoring throughout the entire distribution network.

The decentralization of a power network, actually the smart grid design, covers several disciplines such as (1) electricity, there are multiple power sources using different technologies; (2) networking, there must exist a secure communication between all the nodes which generate data on the system; (3) computer engineering, in the sense that this data must be stored and computed.

The purpose of this paper is to focus on the computer engineering field and propose an architecture and its storage protocol, able to efficiently store and ease the computation of any data generated by the power network inspired by the flavors of cloud computing. This distributed storage architecture is slightly different than the ones used on web services [27] or in pure cloud computing based storage architectures [34, 25] since smart grids

*The research leading to these results has received funding from the European Union European Atomic Energy Community Seventh Framework Programme (FP7/2007-2013 FP7/2007-2011) under grant agreement nr. 247938 for Joan Navarro and August Climent and by the Spanish National Science Foundation (MEC) (grant TIN2009-14460-C03-02) for José Enrique Armendáriz-Iñigo.

†Distributed Systems Research Group, La Salle - Ramon Llull University, 08022 Barcelona, Spain (jnavarro@salle.url.edu).

‡Dpto. Ing. Matemática e Informática, Universidad Pública de Navarra, 31006 Pamplona, Spain (enrique.armendariz@unavarra.es).

§Distributed Systems Research Group, La Salle - Ramon Llull University, 08022 Barcelona, Spain (augc@salle.url.edu).

demand a set of requirements that have not been explored yet. Hence, we also provide a rough correctness verification of the distributed storage protocol.

The reminder of this paper is organized as follows: Section 2 describes the requirements that the distributed storage system must fulfill within the power smart grid framework. Section 3 describes the proposed architecture and explains how it has to be included within the smart grid. Section 4 reviews the system correctness in absence of failures. Section 5 discusses our proposal and suggests other domains of application. Finally, Section 6 concludes the paper.

2. Smart Grids Storage Requirements. Smart grids, as opposite to classical power networks, have become data driven applications since they own a management layer which takes decisions based on the current status of the network. This issue forces designers to redefine the whole power network architecture and its specifications, as now there is a need for storing and processing these datum besides supplying power. This section reviews such requirements and states the basics of the architecture proposed in Section 3.

The strongest requirements of any device inside a smart grid are availability and reliability since denials of service are not acceptable at any situation. Moreover, smart grids are demanded to perform many computations from data collected by smart meters and intelligent electronic devices (e.g. circuit breakers, voltage regulators, etc.). This, extends both requirements—availability and reliability—not only to the physical infrastructure, targeted at supplying energy, but also to datum and their storage.

To fulfill these constraints, we propose a distributed storage architecture built on top of the power smart network able to afford the dynamic behavior of smart grids (e.g. a solar panel may stop supplying energy or an end-user consumer may switch from its local power generation device to the supplier generation network). Actually, distributed storage systems are known to provide availability, reliability, and fault tolerance on many scenarios [15, 17].

Distributed storage systems can be either static or dynamic. Static systems [17, 26] require to know the identity of all nodes a priori in order to be able to distribute storage and computation. On the contrary, dynamic systems [2, 34, 22, 25, 13] do not make any assumption about the system composition, which allow processes joining and leaving at will. This kind of systems are designed to be fault tolerant, understood as the ability to tolerate erratic behaviors from random nodes, and used to improve the scalability of static systems by relaxing the data consistency restrictions [7].

Smart grids demand a trade-off between both scenarios because they behave as a dynamic system but they may need some strong consistency [33] requirements that typical cloud based techniques are currently unable to offer. Hence, our proposal is to build a hybrid system which takes advantage of both distributed storage system schemes, static and dynamic.

Far from just storing data, there are also several applications (also referred to as smart functions) that must run over the smart grid, such as power flow monitoring, under/over voltage monitoring, load shedding, or fault analysis. Each application has its own particular requirements so the proposed architecture must be flexible enough to support such variety of functions. Thus, the distributed storage architecture must provide the following:

1. Reliability. The proposed architecture must be fully tested since major changes on it may imply eventual denial of services.
2. Availability. The architecture must ensure that there always be available data despite its level of consistency.
3. Fault tolerance and recovery. The system must be able to reconfigure its internal characteristics in order to keep supplying and storing data in case of failure.
4. Dynamic consistency. Smart grids run several functions that might require different levels of consistency. For example, on one hand, data needed to perform a load shedding requires strong consistency [33] since it performs critical operations with the current values of the network. On the other hand, data needed to perform power monitoring might require a weaker level of consistency since this function tolerates some kind of delay.
5. Minimum message exchange. It is important to keep a low network overhead in order to guarantee that there were no bottlenecks, and data will flow over the network in an efficient way.
6. Simplicity. This is the key to build a system easy to maintain and adaptable to other domains of application.

The next section proposes the distributed storage architecture and places it inside the smart grid.

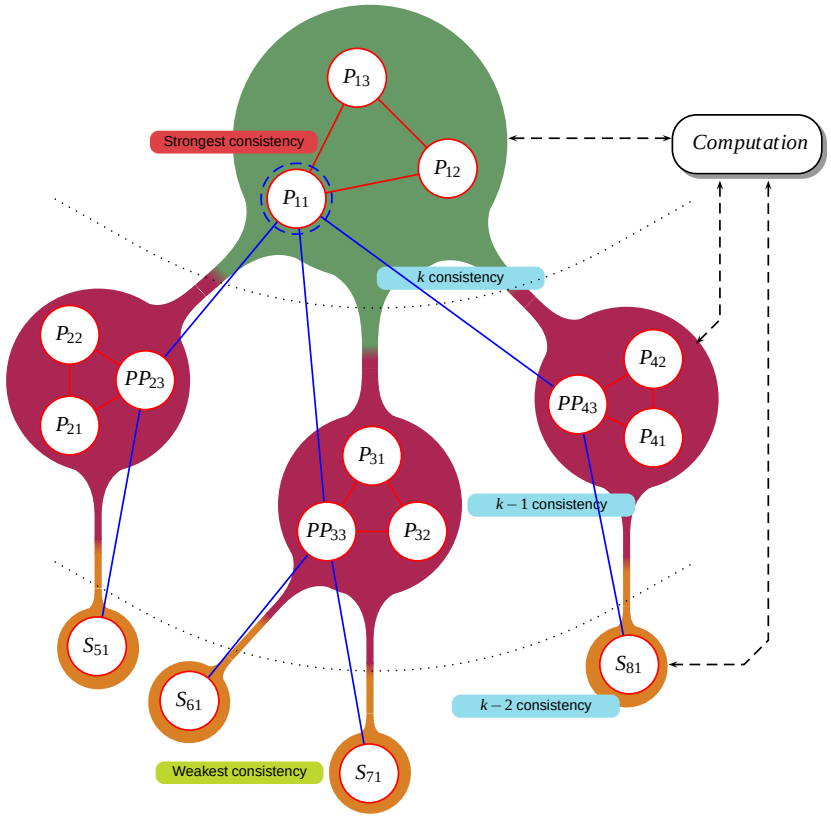

Fig. 3.1: Proposed distributed storage system.

3. System Architecture. Our proposal is inspired on the Primary Copy replication technique [35] used in transactional environments, where all updates are handled by a single node (also referred to as primary) and it propagates them to their replicas (also referred to as subscribers). However, our approach is considerably different from the classical primary copy protocol in terms of (1) primary updates, (2) subscriber updates, and (3) scalability.

Our architecture supports several primary nodes; actually each node can be considered as a primary which means that now updates' load is balanced between several devices. Each primary node treats the rest of devices in the smart grid as subscribers. In addition, an epidemic updates based protocol is proposed in order to replicate data across subscribers. As shown below, in order to overcome the classical scalability constraints of transactional systems our architecture breaks up with the relational scheme and deals with datum as key-value pairs.

This section details thoroughly these three key points and fits the replication protocol on top of the power smart grid infrastructure. Hence, it does not depend on any low layer hardware nor software specification.

As depicted in Fig. 3.1 a smart grid is seen as a set of intelligent electronic device clusters linked by a telecommunications network (i.e. power line communication, wireless network, wide area networks, etc.). A cluster is composed of up to ten devices (drawn as rounded circles in Fig. 3.1) placed on the same geographical area. Each device has limited storage and computing capabilities since it might not be able to solve the whole required smart functions on its own. We also consider that smart meters are attached to these devices and rely on them to report their measurements to the rest of the power smart grid through the computation unit.

Each device in the cluster is labeled as X_{ij} where X corresponds to the device role in the cluster (**P**rimary, **P**seudo-**P**rimary or **S**econdary); i is the cluster identifier, and j is the device identifier. In the same way, we find very useful to define the ancestor of a cluster$_m$ as the node X_{ij} (that obviously belongs to cluster$_i$ $(m \neq i)$) which is updating an arbitrary pseudo-primary k of this cluster (PP_{mk}). Fig. 3.1 shows an example where we have that region 2 is formed by devices $P_{2k} \mid k = [1, 3]$ where P_{21} is the primary of this region; P_{22} is a common

device; P_{23} is the pseudo-primary, (that's why it is named $\mathbf{PP}_{23}$); and, its ancestor is P_{11}. Respectively, S_{61} is the only device on region 6 and its ancestor is PP_{33}.

Regarding data consistency, we define the replication depth r as the amount of different clusters that data are allowed to cross when they are being replicated. This value might be dynamically tuned according to the computation latency or the system performance.

Next, we describe the proposed architecture and explain how we solve the replication, consistency, and fault tolerance issues.

3.1. Architecture Overview. Although the number of smart sensors may substantially increase as time goes by, the number of devices that control them should not grow in the same way. The proposed architecture focuses on the devices instead of the smart meters which is an attempt to avoid scalability issues from the latter ones by hiding their dynamism. However, the system must be robust against possible node failures which forces designers to implement some techniques commonly used in dynamic systems [15, 13, 2].

In order to provide a high available system able to ease the smart functions distributed computation, we propose an architecture inspired on the Primary Copy [17, 35, 30] (also referred to as Primary Backup) scheme.

We distinguish two different types of smart clusters: (1) storage clusters which do not generate data (regions 5, 6, 7, and 8 in Fig. 3.1) and (2) active clusters that generate data (regions 1, 2, 3, and 4 in Fig. 3.1).

Any device belonging to an active cluster may simultaneously adopt different roles according to the current situation: (1) primary master, (2) primary slave, (3) pseudo-primary, and (4) secondary. When a device is propagating data from their directly attached smart meters, it will act as a primary master and will treat the rest of devices in its cluster as their primary slaves (in Fig. 3.1, P_{11}, marked with a dashed blue circle, is the primary master and P_{12}, P_{13} are their primary slaves). When a device receives data from another cluster it will be acting as a repeater (pseudo-primary) (in Fig. 3.1 PP_{23}, PP_{33}, and PP_{43} are the pseudo-primaries of P_{11}). Moreover, if a device receives updates from other clusters but it does not propagate them, it will be acting as a secondary (in Fig. 3.1, $S_{51}, S_{61}, S_{71}, S_{81}$ behave as secondary devices).

Note that blue lines in Fig. 3.1 just illustrate the particular case of P_{11} broadcasting data. In fact, the proposed architecture must be understood as if all nodes where generating data. Hence all nodes may act as primary, pseudo-primary, and secondary devices at a time.

Smart grids need to compute many smart functions [10] indeed. Our architecture is flexible enough and able to adapt itself to the data freshness requirements [30, 4] imposed by each smart function. This way of propagating updates along the replication tree structure allows the system to find the most appropriate version for computing a function while circumventing all traditional problems of scalability and availability in these approaches [26, 4, 19, 35]; this will be described in a more detailed manner in Section 3.2. Hence, the primary master of a region (i.e., P_{11}) must decide when to passively replicate its data to the pseudo-primaries of the neighboring regions (P_{21}, P_{31}, P_{41}). At the same time, each pseudo-primary has to take the same decision with its data and their pseudo-primaries or secondaries. These decisions must be taken according to (1) the function periodicity (i.e., flow monitoring will require faster updates than asset management), (2) link status and congestion, and (3) cluster status (i.e., a general master might decide to asynchronously replicate its data when it detects that there are very few alive nodes on its cluster).

Actually, once the primary master has sent its data to a pseudo-primary node of another cluster, a recursive process starts where each pseudo-primary looks for another pseudo-primary in another neighboring cluster to propagate its data. This process finishes when there are no more clusters or there is a cluster which has no more neighbors (i.e., S_{51}, S_{61}, S_{71}, and S_{81}). The decision process must be aware of not falling in cluster loops and ensure that in each cluster there is only one pseudo-primary node that contains data from the primary master.

Each time the computation unit of the smart grid needs to calculate the result of a given smart function, it first attempts to use data from its nearest neighboring cluster. If data contained on that cluster has a consistency level k greater than l, where l is the freshness level required by the function, it will use that cluster to perform computation. Otherwise, it will get redirected to its ancestor node with a freshness index $k-1$ and repeat the operation.

The following subsections detail how our architecture deals with replication, consistency and fault tolerance issues.

3.2. Replication. Distributed systems replicate data to provide scalability, availability, and fault tolerance. However, replication increases the number of messages in the sense that all replicas have to be synchronized

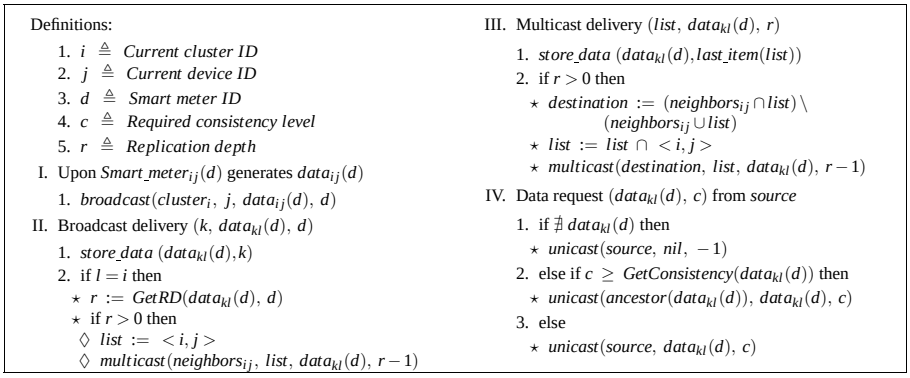

Fig. 3.2: Replication protocol at smart device$_{ij}$

which can potentially reduce the system throughput. To avoid this situation several techniques are proposed in the literature [31, 15, 11, 2].

Regarding the time when updates get propagated to the replicas there exist two major strategies: eager and lazy replication. On one hand, eager replication [5] consists of writing data to all replicas before finishing the write operation (similar to 2PC in databases). This solution provides strong consistency [33] but has limited scalability. On the other hand, lazy replication [21, 35] consists of writing to all replicas after exhausting the write operation. This technique achieves higher scalability but has more difficulties to maintain consistency—i.e. replicas may diverge.

Regarding the amount of replicas that update data, there exist two major strategies: active and passive replication. Active replication [3] broadcasts updates to all replicas at a time. Again, this technique provides strong consistency since all replicas are easily synchronized but has low scalability. Passive replication [29] processes updates on a single site called *primary* which propagates its updates to another site called *backup*. At the same time, this backup site can also propagate its state to another backup site until achieving the desired replication depth. This solution provides higher scalability but has some troubles on maintaining strong consistency since all replicas might be unsynchronized.

The final consideration regarding to scalability is the number of messages exchanged [36]; this is a critical factor too since the greater the number of messages exchanged per operation are the more network stalls we have. We can consider that there can be a linear interaction where the number of messages exchanged depends on the kind of operation; or, otherwise, constant interaction where a fixed number of messages is exchanged per operation. The former features poor scalability while the latter one can increase the scalability though we have to take into account that they have to be kept to a minimum (ideally one message and, at most, two).

In our system we have taken these previous considerations and propose a hybrid solution. We have considered to mix passive and active replication; however, we consider the propagation of changes to a small subset of replicas and the replicas belonging to this subset are responsible for propagating the changes to another disjoint small subset of replicas and so on up to the replication depth specified by the system for a certain variable. As depicted in Fig. 3.2, our proposal is a hybrid solution between all these techniques and benefits from the strengths of each solution:

1. When a device (primary master, P_{11} in Fig. 3.1) receives data from its smart meters it eagerly replicates them by *broadcast*ing (step I in Fig. 3.2) these data to all devices within its cluster (primary slaves, i.e., P_{12} and P_{13} in Fig. 3.1). Therefore, it is performing an active replication (step II.1 in Fig. 3.2).
2. If the replication depth r associated to these kind of data is greater than 0, the primary master has also to lazily replicate these data to other clusters (1) avoiding replication loops and (2) *multicast*ing relevant meta-data.
   In order to avoid replication loops (i.e., the same cluster has different versions of the same object), the device must build a *list* with the ancestors of data it is currently processing and remove all devices of its neighbor list that are in the ancestor list. Recall that each primary, or pseudo-primary, device has only one pseudo-primary per region.
   Regarding the neighbor discovery, we assume that given a neighboring cluster B from cluster A, all the

nodes in cluster A will choose the same pseudo-primary from cluster B. If this pseudo-primary fails, the next pseudo-primary chosen will be the one with the lowest identifier in cluster B. For example, in Fig. 3.1, if P_{33} fails, P_{32} and P_{33} will also chose S_{61} and S_{71} as their pseudo-primaries.
Upon the neighbor list has been pruned (and updated), the device *multicast*s (1) the ancestors *list*, (2) the stored $data_{kl}(d)$ and (3) a decremented value of replication depth to its neighbors (i.e., PP_{23}, PP_{33} and PP_{43} in Fig. 3.1) as shown in step II.2 in Fig. 3.2.

3. This is repeated while the replication depth is greater than 0 as shown in step III in Fig. 3.2. Note that we are actually performing active replication within devices of primaries' area and passive replication within devices of different clusters.
The first time data from $smart\ meter_d$ achieves the latest pseudo-primary (r has reached the 0 value), or secondary (i.e. $S_{51}, S_{61}, S_{71}, S_{81}$ in Fig. 3.1) this device will send to the computation unit its identifier. This ensures that the computation unit will eventually know where to find the furthest replica of data associated to $smart\ meter_d$. Recall that the nearest device that contains these data is its primary, which is also known by the computation unit in advance.
4. When a device receives a data query from the computation unit, it first checks if the consistency level of its stored data is enough to perform computation. If it is greater than the one required c, it will give back its stored data, otherwise it will forward this query to its ancestor as shown in step IV in Fig. 3.2. This solution is inspired in the passive replication technique and eases distributed computation as discussed in Section 5.

Therefore, this replication protocol performs both (1) active and eager replication in the primary-master's cluster and (2) passive lazily replication in other clusters inspired in the fundamentals of a primary-copy scheme. This improves the scalability issues of classical architectures [17, 19] since we have restricted an controlled the amount of replicas that receive the changes at a time and implement different regions (set of clusters) with different consistency levels. This last feature is very interesting; some computations do not need to be exclusively forwarded to the primary replica, it can be delegated to other replicas and, thus, the system achieves a better performance. This can be best seen with two examples, if we want to check the consumption of a subscriber for billing purposes then we will need to go to the primary (strong consistency). The other example can consist in checking the power consumption of a certain urban area to detect the variation and decide whether to derive or not more power to that area (this is specially useful in summer with air conditioning systems) and the consistency is not so critical.

Finally, we define the *replication chain* as the closed set of devices which exchange versions of the same data item. For example, in Fig. 3.1 there are four replication chains concerning data generated by smart meters attached to P_{11}: $\{P_{11}, PP_{23}, S_{51}\}$, $\{P_{11}, PP_{33}, S_{61}\}$, $\{P_{11}, PP_{33}, S_{71}\}$, and $\{P_{11}, PP_{43}, S_{81}\}$.

The following section describes how consistency is kept under this replication protocol.

3.3. Consistency. Research on consistency protocols has been conducted for many years and several approaches have been proposed by the community. First attempts [5] on keeping consistency in distributed systems consisted of building serializable plans by avoiding read and write operations performed concurrently over the same data item. This technique ensures that data will be always seen by anyone immediately after its update; which is also known as strong consistency [33]. Later, as the performance requirements increased, researchers relaxed the consistency guarantees by defining the weak consistency [33] with the aim of improving scalability and availability. These techniques [33] accept a limited period of time (also referred to as inconsistency window) where updates are available only to a subset of sites in the system. When the inconsistency window expires, they ensure that data is consistent.

Generally, this leads us to two major alternatives when defining the consistency properties of a system: (1) strong consistency and (2) weak consistency. Strong consistency provides us with high consistent systems but with poor scalability and availability since all replicas must be synchronized. In the other hand, weak consistency provides higher scalability and availability but limited consistency features.

Regarding our proposal, we take advantage of both strong and weak consistency strategies and propose a hybrid solution inspired by cloud-based storage and data stream warehouses [15, 11, 16].

Cloud computing-based storage techniques [34, 25], on its effort to achieve high scalability and availability, typically implement a specific form of weak consistency: eventual consistency [33]. This technique states that once data is updated, all replicas will eventually get that value if no more updates are applied.

Smart grids' behavior has many similarities with stream warehouses [16] since there is a continuous stream

of data (generated by smart meters) which has to be stored. In order to overcome the classical update and query consistency issues given on such scenarios [16], our architecture guarantees that there will never exist multiple attempts to write (or read) the same data item into the same place. Note that two or more replication chains of the same data item never converge to the same cluster. Nevertheless, we find very useful to use multi-version [6] techniques used in stream warehouses to maintain a notion of consistency between different data stored in each device and enhance the global system performance.

Although IEC 61850 standard [20] defines the data model that a smart grid should store and our architecture deals with this model by hiding data structures inside devices, we have to take care about data periodicity. Each smart meter generates data at different intervals, i.e., voltage measurements might be generated with a higher frequency than heat measurements. Hence, each data measurement is stored with the time stamp k it was originally acquired, similar to the version technique used in stream warehouses [16, 6].

Actually, there exists a close relationship between this time stamp k in which data are generated and the consistency of these data. Roughly speaking we can assure that the greater the k, the more recent data measurement is but its consistency is potentially weaker. Recall that here we understand consistency as the property which states that all the members in the replication chain own the same data item with time stamp k (also referred to as version).

In fact, when our architecture is required to store stream data measurements [16, 33], the most recent data versions will be always located closer to their sources; whereas oldest versions might have already reached the furthest devices in the replication chain. However, if a given measurement is not so frequently taken, then the most recent version will be found anywhere in its associated replication chain.

In some use cases, the computation unit can still work with older versions (weakly consistent) to perform the calculations required. Hence, when it wants to obtain a specific data item, it can include a k value to state that the computation should be done with values that equal to (or greater than) k. To this end, queries will be traveling along the replication chain associated to each data measurement to find the proper version up to its master (in the case that the required k version is not found).

In the master's cluster we implement strong consistency between all replicas (P_{11}, P_{12} and P_{13} in Fig. 3.1). This improves the fault tolerance of the system since another device (primary-slave) of the cluster could easily take over from a primary-master's fault. Moreover, this keeps us safe from the typical single point of failure problem. Actually, all data stored in any device belonging to the master's cluster has the highest k level of consistency, since no newest data have been generated.

Once data are strongly consistent in the master's cluster, devices start propagating them with a time stamp k to their pseudo-primaries (as shown in Fig. 3.2, P_{11} will broadcast to PP_{23}, PP_{33} and PP_{43}). Recall that decisions of when data must be propagated are taken according to the smart functions and system status. Therefore, we are currently implementing an eventually consistent system between the pseudo-primaries. It is more likely that data stored in these pseudo-primaries will provide weak consistency since new data measurements will come from smart meters with a time stamp strictly greater than k.

To sum up, from the consistency point of view, we have shown how our hybrid architecture uses both strong and weak (actually k-weak) consistency techniques. Next we describe how our architecture deals with fault tolerance.

3.4. Fault Tolerance. Fault tolerance is understood as the ability of the system to recover from a spontaneous site fault. Distributed systems are prone to different types of site failures [12]. Researchers have to adapt their techniques according to the domain of use and the intrinsic characteristics of the distributed system. This section explores (1) which kind of failures smart grids are prone and (2) how our proposal acts against them.

Since smart grids are hardly dependent on the communication network, we can assume that this channel will be reliable enough and focus our efforts on the distributed storage architecture. Therefore, our goal is to implement such a policy that in case of a node failure, the global system could still behave properly. In this case, we assume that any site may fail according to the crash model [12].

If a server started behaving in an arbitrary manner (also referred to as byzantine model), it would be either because it is returning or propagating an arbitrary or older (though valid) version of a variable. We control this by adding a digest to the value stored, similar to what it has been proposed in [9, 28]. Whenever we found a mismatch between them we would force the replica to shut down.

Regarding our proposal, there may exist two different failure cases. The first one corresponds to the failure of a primary (i.e., P_{11} in Fig. 3.1) and the second one to the failure of a pseudo-primary (i.e., PP_{23} in Fig.

3.1). In the former, we inherit the advantages of the active replication techniques and are able to recover easily: when the primary master fails (e.g., P_{11} in Fig. 3.1), any other primary-slave can immediately take over the situation. Recall that due to eager replication, they are completely sharing the state of the primary master.

In the latter failure scenario, any of the nodes of the cluster where the failure takes place can become the new pseudo-primary and continue with data transmission and replication. Unfortunately, due to the fact that pseudo-primaries perform passive and lazy replication, the takeover process is not as fast as in the primary's cluster. As soon as the ancestor, belonging to cluster A, of the failed node, belonging to cluster B, detects its unresponsiveness, it will select a new pseudo-primary from cluster B. If no more pseudo-primaries (or secondaries) are available (i.e., cluster 7 in Fig. 3.1) it will send a message to the computation unit informing that it is the last device of the replication chain. Otherwise, thanks to the neighbor discovery function, it can easily find its successor in the replication tree and continue with the replication of the data in the system.

The challenge here is that the takeover solution in a pseudo-primary implies that the previous versions of a given data item are lost. We have to define a state transfer protocol so that every cluster contains the most complete and up-to-date information. Otherwise, there might exist certain network clusters where we cannot achieve the desired degree of consistency due to the fact that these *new* pseudo-primaries do not completely store the state. In Section 5, we will deal with this data transfer that can range from a full state transfer to nothing sent. In all this range, we are going to see the advantages and disadvantages of each solution.

However, this is not enough since most of these processes do not completely stop forever after their failure. This system has certain dynamism, in the sense that components can be repaired. Hence, there can be a role re-assignment and the need of recovery tasks for previously crashed nodes.

The following subsection proposes an analytical study of the scale out factor of our protocol in order to provide with some arguments to warrant the viability of our approach.

3.5. Proof of concept. As one of the major goals of our proposal is achieving high scalability, we use the analytical model and notation from [31] which estimates the scale out factor in a replicated database system when there is an increase of the number of sites and replicas. Thus, we briefly describe the adaption to our system model used to proceed with the computation of the scale out factor.

The scale out factor determines how the performance of the global system is increased or decreased by using replication. As shown in Equation 3.1, this is computed as the sum of work executed at each replica divided by the processing capacity of a non-replicated database.

$$Scale\ \ Out = \frac{1}{C} \sum_{i=1}^{n} \sum_{j=1}^{m} C \cdot U_{ij} \cdot ACC_{ij} \quad [31]. \tag{3.1}$$

From the previous equation, we have that C is the processing capacity of a non-replicated database, n is the number of replicas in our smart grid, m is the number of stored objects, U_{ij} defines the location of object j at site i and ACC_{ij} defines the accessibility of object j at site i. Hence, if the object j is at site i, then $U_{ij} = 1$ which defines the replication schema. Actually, ACC_{ij} defines the access rate to object j at site i.

However, Equation 3.1 assumes that read and update operations launched against the replicated distributed system are uniformly spread across all replicas. Hence, this equation is not directly applicable to our proposal because our solution has an in-built load balancing mechanism which redirects operations to replicas according to the required consistency level k. Thus, we show how we have adapted the analytical description of the replicated system to fit in our system characteristics.

Recall that the term $(C \cdot U_{ij} ACC_{ij})$ is equivalent to $(C_r \cdot ACCR_{ij} + C_u \cdot ACCU_{ij})$ where C_r is the read processing capacity, C_u is the update processing capacity, $ACCR_{ij}$ is the read accessibility and $ACCU_{ij}$ is the update accessibility. So, replacing this expression in Equation 3.1 we obtain Equation 3.2 which is now useful in our replication protocol.

$$Scale\ \ Out = \frac{1}{C} \sum_{i=1}^{n} \sum_{j=1}^{m} C_r \cdot ACCR_{ij} + C_u \cdot ACCU_{ij}. \tag{3.2}$$

Actually, Equation 3.1 can be considered a generalization of 3.2. We now evaluate the different replication strategies for a system from 10 to 80 sites, 10 objects, 80% of read operations and 20% of write operations to

Table 3.1: Scale out evaluation

Sites	Full Replication	Partial Replication	**Smart Grid**
0	0	0	0
10	10	9,8	9,82
20	15,7	16,9	18,15
40	25	28,2	30,7
80	40	48,2	53

all objects. The distribution of objects and the operations on objects are evenly distributed. We also assume that all sites have the same processing capacity (i.e. all smart meters have similar specifications and storage capabilities).

We have compared the scale out factor obtained in our proposed protocol against the replication strategies proposed in [31]: (1) full replication—all sites contain the same data—and (2) partial replication —a reduced set of sites contains a given data. In the case of the latter, we have chosen to replicate each data item in $n/2$ sites in order to obtain comparative results. The scale out evaluation of these protocols is shown in Table 3.1.

Ideally, the scale out factor should be equal to the number of sites of the system which meant that all incoming updates are being processed without saturation. We can see that with a full replication scheme the scale out is quite poor: 40 with 80 sites. When the number of sites increases, the system cannot scale anymore as the full replication policy forces that all updates have to be sent to all replicas which takes a considerable amount of time.

In contrast, with the partial replication (limited to half of the replicas) the system scales slightly better because the cost of propagating the replicas is not so high (scale out of $48,2$ with 80 sites). Finally, our cloud based replication protocol scales out is even better (53 at 80 sites). In this case the replicas that have the most recent data version have a higher C_u and lower $ACCR_{ij}$ because most read operations are executed in replicas that do not have necessary the last version of the data. For these reasons, we show that the scale out is better than traditional full replication and partial database replication protocols. This proof of concept makes us believe that this is a good approach in power smart grids.

This subsection finishes the description of our proposal. In the following section, we roughly verify the correctness properties of this system if all nodes behave properly and faults never occur.

4. Correctness Guarantees. This section provides arguments for correctness of the global distributed system in a failure free environment. Distributed algorithms are said to be formally correct when their liveness and safety properties are satisfied and shown to be correct. Regarding the liveness property, it can be best seen as something good will eventually happen; while, the safety property can be stated as nothing bad will happen.

Our proposal needs to propagate the measures from a given smart meter from zone to zone up to its replication level; hence, changes need to get propagated and applied in the same order in all pseudo-primaries. Both asserts constitute the liveness and safety properties of our system.

Next, we point out some guarantees of the system extracted from the previous sections in order to have enough arguments to warrant the correctness properties.

Guarantee 1. Any primary (or pseudo-primary) will never send the same data item to more than one device per neighboring cluster.
This is guaranteed since data ancestors are excluded from the device's neighbor list as shown in step III.2 in Fig. 3.2.

Guarantee 2. The computation unit knows which is the last pseudo-primary (or secondary) of the replication chain.
As described in section 3.2, when a device notices that a new data item has gone through all its replication depth ($r = 0$), it will send a message to the computation unit identifying itself.

Guarantee 3. Any device belonging to a master cluster has always the latest version of data generated by any smart meter within that cluster.
This is satisfied since data is eagerly replicated to all devices of the master cluster as shown in steps I.1 and II.1 in Fig. 3.2.

From this guarantees, we can state the safety and liveness properties of our system. As the system behaves

different depending if it is replicating data (also referred to as write) or executing a query from the computation unit (also referred to as read), both properties (safety and liveness) must be analyzed in two facets: read and write.

4.1. Safety Properties. The safety properties of our architecture are stated by the following claims:

CLAIM 1. **Safety on write.** Safety on write operations is guaranteed since there will never occur a situation where the same data is being written from two or more different sources.
This is guaranteed since (1) there is only one $smart\_meter_d$ generating $data_d$, (2) point to point communication channels do not disorder messages, and (3) the *neighbor* function will never find more than one device per cluster as stated in Guarantee 1.

CLAIM 2. **Safety on read.** There will always exist a consistent version of the requested data item queried by the computation unit.
This is assured provided that the replication protocol (Fig. 3.2) guarantees consistent writes throughout the whole replication chain.

4.2. Liveness Properties. The liveness properties of our architecture are stated by the following claims:

CLAIM 3. **Liveness on write.** Liveness on data updates is trivially assured by Guarantee 1.
Data generated on the smart meter will follow the replication chain and being consistently written at each device until the replication depth reaches a value of 0 or there are no more neighbors.

CLAIM 4. **Liveness on read.** Liveness on data reads is guaranteed provided that Claim 2 is accomplished.
The computation unit will always send queries to the last device of the replication chain. If data contained in it have not the required consistency level k, the device will redirect the query to its ancestor. This will happen in a recursive way until the required level k is found. If any device can reach the required level k the primary will give back with its latest version which is strongly consistent as provided by Guarantee 3.

Finally, in the following section we discuss some limitations of our approach.

5. Discussion. As shown in the previous sections, our proposal takes benefit from many techniques used in distributed systems. However, to the best of our knowledge, these techniques have never been put together nor tested. Hence, there are several aspects that have been intentionally left out and need to be discussed:

Master cluster reduction. Not all the members of the master region have to participate on active replication. Along this work, we have assumed that all nodes of a given primary cluster participate in the active replication of all data. In fact, this is not necessary at all: although the number of nodes belonging to a zone can be in the range of tens, we think that we can speed up the replication process by selecting a set of representatives for each subset of smart meters. It is well known that active replication does not scale well [35] and with the proper selection of representatives the rest can become secondaries of each representative.

Enhancing the takeover process. A pseudo-primary (PP) could do active replication within its associated cluster. This role is not an exclusive one in the cluster, it can be also responsible for several smart meters and, thus, collaborate in the active replication protocol.

Failure detection. In Section 3.4 we have stated that the ancestor detects the failure of its successor in the replication hierarchy; this can be achieved by implementing a timeout plicy. However, the update propagation frequency may vary since data to be transmitted is very different (i.e., monthly power consumption is less frequent than voltage monitoring at a given point in the network).

Hence, it makes sense to think that inside the cluster of a given pseudo-primary (or secondary) the active replication among their nodes would detect the failure of the pseudo-primary (or another device). If so, they can agree with selecting a new pseudo primary in that cluster and notify the ancestor about this fact. Again, we have to reconstruct the new hierarchy tree in order to add the new pseudo-primary. However, this would overload the pseudo-primaries' clusters and might worsen the global performance of the system.

State transfer in presence of faults. It is said, that when a pseudo-primary fails we need to transfer its data (missing data) to the new pseudo-primary as its copy is lost once it fails. We can have several alternatives to mitigate this effect as this system is derived from a data driven application.

On one hand, we can perform an active replication of each pseudo-primary in its cluster zone. As mentioned in the previous point, this approach might not be feasible since we increase the load of the system as we go deep in the replication structure of the system, which potentially may flood the whole network; leading with a not desirable situation. Note that this solution does not need to transfer any state information from anywhere in case of fault.

On the other hand, we have the alternative to transfer the full state to the new successor from the ancestor (recall that it belongs to another cluster and this could be costly); however, this alternative has several drawbacks. The first one is that it might affect the availability of the system since transferring the whole data may affect the transmission of new data to the successors. The second one is that the volume to transfer might be so big that it could not catch up with the current state of the system [32].

There may exist an hybrid solution: we could perform a partial state transfer of data. This implies that the replication algorithm has to ensure that each pseudo primary has a set of secondaries in its associated cluster where the updates get also propagated asynchronously. Therefore, when a given pseudo-primary fails, it is only needed to transfer a much less amount of data to the new pseudo-primary than in the case of full state transfer. Nevertheless, this has to be tested and checked in order to find out the proper number of pseudo-primary slaves and the amount of data transferred per round.

Distributed computing. Our proposed architecture allows to perform distributed computation on the read steps. Thanks to the fact that required data travel across the replication chain (depending on the required consistency level k) each node might be able perform a piece of the computation required.

Actually, as suggested in [14] for cloud computing environments, with our architecture it would be possible to implement something similar to MapReduce; where the node owning the required data version run the *map* tasks and the rest of nodes in the replication chain continuously run the *reduce* tasks. Such distributed computation not only might reduce the size of the traveling data but also improve the computation throughput of the smart grid.

However, we have not considered this feature in this work but it can be seen as a chance to improve the intelligence and power of the system.

Dynamic replication depth tuning. We believe that if we were able to dynamically adjust this value our system might adapt better to their requirements. In fact, we have not specifically stated how the replication depth is set, neither augmented or decreased. It can be adjusted by the system administrator but it can also be dynamically adjusted as a function of the demands coming from the computation unit. Moreover, it can be tuned autonomously in case of disaster or rapid evaluation of certain functions (e.g. accounting). Further, there might exist certain information that would need to be replicated in all nodes as it is rarely modified.

To this end, we are thinking about a cognitive system [18] based on supervised learning in order to (1) evaluate the whole system status and (2) predict the optimal value of the replication depth for each data item. Actually, we are implementing a learning classifier system (e.g., XCS or UCS) [37] able to adapt itself to the system dynamism due its online nature.

6. Conclusions and Future Work. This paper presents a novelty approach to take advantage of smart electric grids. We have defined a way to distribute and store information across the network so that the computation needed for certain smart functions can be greatly reduced. We have detailed the replication protocol based on epidemic updates and proofed its correctness. This work aims to provide some insight into the world of smart grids from a data perspective. For the sake of simplicity during the presentation of our system, we have outlined simple scenarios about the replication policy or fault-tolerance issues that need to be treated in detail in further works.

In addition, future work should be targeted at (1) implementing the architecture in a real-world scenario to obtain numerical values of its performance, and (2) defining the cognitive system behavior in order to tune and optimize the replication protocol according to real data access and update patterns.

REFERENCES

[1] AEP Ohio. About the gridsmart project, Feb 2011. Available in: `www.aepohio.com/save/demoproject/about/Default.aspx`.
[2] Marcos Kawazoe Aguilera and et al. Sinfonia: A new paradigm for building scalable distributed systems. *ACM Trans. Comput. Syst.*, 27(3), 2009.
[3] Yair Amir and Ciprian Tutu. From total order to database replication. In *ICDCS*, pages 494–, 2002.
[4] José Enrique Armendáriz-Iñigo, J. R. Juárez-Rodríguez, José Ramón González de Mendívil, Hendrik Decker, and Francesc D. Muñoz-Escoí. *k*-bound GSI: a flexible database replication protocol. In Yookun Cho, Roger L. Wainwright, Hisham Haddad, Sung Y. Shin, and Yong Wan Koo, editors, *SAC*, pages 556–560. ACM, 2007.
[5] Philip A. Bernstein and et al. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1987.
[6] Irina Botan and et al. Secret: A model for analysis of the execution semantics of stream processing systems. *PVLDB*, 3(1):232–243, 2010.
[7] Eric A. Brewer. Towards robust distributed systems. In *PODC Conf.*, NY, USA, 2000. ACM.

[8] Richard E. Brown. Impact of Smart Grid on distribution system design. In *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, pages 1–4, 2008.
[9] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
[10] A. Chuang and M. McGranaghan. Functions of a local controller to coordinate distributed resources in a smart grid. In *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, pages 1 –6, 2008.
[11] Brian F. Cooper and et al. Pnuts: Yahoo!'s hosted data serving platform. *PVLDB*, 1(2):1277–1288, 2008.
[12] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78, 1991.
[13] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Elastras: An elastic transactional data store in the cloud. *CoRR*, abs/1008.3751, 2010.
[14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010.
[15] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, pages 205–220, 2007.
[16] Lukasz Golab and Theodore Johnson. Consistency in a Stream Warehouse. In *CIDR*, 2011.
[17] Jim Gray and et al. The dangers of replication and a solution. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 173–182. ACM Press, 1996.
[18] J. Holland. *Adaptation in natural and artificial systems.* The MIT Press, second edition, 1992.
[19] Ricardo Jiménez-Peris, Marta Patiño-Martínez, Bettina Kemme, and Gustavo Alonso. Improving the scalability of fault-tolerant database clusters. In *ICDCS*, pages 477–484, 2002.
[20] Tatjana Kostic, Otto Preiss, and Christian Frei. Understanding and using the iec 61850: a case for meta-modelling. *Computer Standards & Interfaces*, 27(6):679–695, 2005.
[21] Konstantinos Krikellas, Sameh Elnikety, Zografoula Vagena, and Orion Hodson. Strongly consistent replication for a bargain. In *ICDE*, pages 52–63, 2010.
[22] Lakshman, Avinash and Malik, Prashant. Cassandra: a decentralized structured storage system. *SIGOPS Operating Systems Review*, 44(2), Apr 2010.
[23] Masdar. Masdar website, Feb 2011. Available in: `www.masdar.ae/en/home/index.aspx`.
[24] mThink. The smart grid in malta, Feb 2011. Available in: `mthink.com/utilities/utilities/smart-grid-malta`.
[25] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon s3 for science grids: a viable solution? In *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64, New York, NY, USA, 2008. ACM.
[26] Marta Patiño-Martínez, Ricardo Jiménez-Peris, Bettina Kemme, and Gustavo Alonso. Middle-r: Consistent database replication at the middleware level. *ACM Trans. Comput. Syst.*, 23(4):375–423, 2005.
[27] Alberto Paz, Francisco Perez-Sorrosal, Marta Patiño-Martínez, and Ricardo Jiménez-Peris. Scalability evaluation of the replication support of jonas, an industrial j2ee application server. In *EDCC*, pages 55–60. IEEE Computer Society, 2010.
[28] Fernando Pedone, Nicolas Schiper, and José Enrique Armendáriz-Iñigo. Byzantine fault-tolerant deferred update replication. In *LADC*, pages 7–16. IEEE Computer Society, 2011.
[29] Fernando Pedone, Matthias Wiesmann, André Schiper, Bettina Kemme, and Gustavo Alonso. Understanding replication in databases and distributed systems. In *ICDCS*, pages 464–474, 2000.
[30] Christian Plattner, Andreas Wapf, and Gustavo Alonso. Searching in time. In *SIGMOD Conference*, pages 754–756, 2006.
[31] Damián Serrano, Marta Patiño-Martínez, Ricardo Jiménez-Peris, and Bettina Kemme. Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. *Pacific rim international symposium on dependable computing, IEEE*, 0:290–297, 2007.
[32] Ricardo Manuel Pereira Vilaça, José Orlando Pereira, Rui Carlos Oliveira, José Enrique Armendáriz-Iñigo, and José Ramón González de Mendívil. On the cost of database clusters reconfiguration. In *SRDS*, pages 259–267, 2009.
[33] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.
[34] White, Tom. *Hadoop: The Definitive Guide.* O'Reilly Media, 1 edition, June 2009.
[35] Matthias Wiesmann and André Schiper. Comparison of database replication techniques based on total order broadcast. *IEEE Trans. Knowl. Data Eng.*, 17(4):551–566, 2005.
[36] Matthias Wiesmann, André Schiper, Fernando Pedone, Bettina Kemme, and Gustavo Alonso. Database replication techniques: A three parameter classification. In *SRDS*, pages 206–215, 2000.
[37] S. Wilson. Classifier fitness based on accuracy. Technical report, The Rowland Institute for Science, 100 Edwin H. Land Blvd. Cambridge, MA 02142, Apr 1995.

Edited by: Dana Petcu and Jose Luis Vazquez-Poletti
Received: August 1, 2011
Accepted: August 31, 2011