*Online Journal of Applied Knowledge Management*
A Publication of the International Institute for Applied Knowledge Management

*Volume 2, Issue 2, 2014*

# Enabling disaster recovery scenario in digital libraries using eventual consistency

**Milorad P. Stević**, milorad.stevic@live.com,The Higher Education Technical School of Professional Studies, Novi Sad, Serbia

**Ivica Ž. Kolenkaš,** ivica.kolenkas@gmail.com,The Higher Education Technical School of Professional Studies, Novi Sad, Serbia

## Abstract

*Knowledge access and knowledge sharing is essential in contemporary classrooms. Building a digital library is important for empowering knowledge access. It is a way to improve efficient knowledge distribution and collaboration among students. Considering the importance of the digital library in the process of knowledge access and knowledge sharing, it is essential to prepare this system for disaster recovery. Disaster recovery includes activities that are related to preparation for natural or human-induced disasters. In a case like this, where the system handles large amounts of digital content, building a disaster recovery scenario is difficult. There are approaches for dealing with this problem that rely on copying data to a distant location, but those are inefficient, slow and they produce system bottlenecks while in action. This paper shows a different approach-proposed architecture depends on the MongoDB NoSQL database system for handling large binary files. If used, MongoDB database system serves as digital library storage, making disaster recovery possible, regardless of the size of the content it manages. It is a distributed database system suitable for handling binary files with advanced replication mechanisms for securing its content.*

**Keywords:** Disaster recovery, Unstructured data types, NoSQL, Service-oriented architecture, Document management, Information systems, Scalability

## Introduction

Digital libraries are unavoidable in modern information systems (IS) in academic institutions. The classic library is not capable of keeping pace with the increasing need for knowledge access and knowledge sharing demands. This further induces changes that the IS needs to conduct, because students are asking for more efficient ways to access and share knowledge. Modern, service-oriented IS in academic institutions should efficiently solve these issues. However, with the growth of demands comes the need for disaster recovery planning which is now much more difficult since the IS has to handle a great amount of digital content. Classic approach with scheduled backups and copy-paste mechanisms cannot cope with the problem anymore. It is usually performed using activities that imply the copying vast amount of files on some distant location. Those processes are time consuming and risky, since stopping the process for any reason often involves repetition/continuation of the procedure, which is inconvenient. On the other hand, not having a disaster recovery scenario is unacceptable, since the risk of data loss is present.

Not having a disaster recovery scenario can lead to serious damages, as reported in India, where one of two university libraries in Punjab state of India lost over 70 percent of its collection in a flood (Kaur, 2009). Developing a disaster recovery scenario related to natural disasters is very

important for academic institutions, as noted in paper that describes the case of the Lamar University in Beaumont, Texas, which was hit twice by hurricanes, from 2005 to 2008 (Beggan, 2011). Sometimes, it is not the technology that resolves issues, but people themselves, using local wisdom-based recovery models as the one in Bantul district, Yogyakarta, Indonesia, following the 2006 earthquake (Kusumasari and Alam, 2012). However, regardless of what type of disaster is under consideration and what kind of disaster recovery scenario is to be conducted, managing resources in disaster recovery projects is essential (Chang et al., 2012).

Managing digital content and developing a disaster recovery scenario for any IS that manages a huge amount of digital content is always connected with expensive hardware needed for the accomplishment of the project. However, even then the process of copying great amount of digital content in a distant location is questionable. It is very important to choose the appropriate model, technique and resources for disaster recovery, since it affects performance and the overall cost of the solution (Guster et al., 2012). Using a relational database management system (RDBMS) for managing unstructured data, as digital content is, is inefficient, because RDBMS were built for managing structured data and have limitations when handling large and numerous unstructured data (Stonebraker et al., 2007). Storing data that is expected to heavily grow inside a RDBMS or on a file system based repositories can be enhanced using vertical scaling, but this approach further burdens the resources (Kossmann, Kraska and Loesing, 2010).

This paper proposes a different approach, which is more architecture-related. Instead of investigating methods for more efficient movement of huge amounts of data, the paper advices that digital content should be handled by some distributed database system, in this case the MongoDB database system. Since MongoDB is a NoSQL database management system, it relies on CAP theorem (C-Consistency; A-Availability; P-partition tolerance), which enables it to run in the eventual consistency model. Because of the CAP theorem, the nature of systems based on NoSQL solutions is efficient handling of unstructured data. The CAP theorem explains possible combinations of three key properties of distributed database systems:

- The system can be consistent and available, but not partition tolerant
- The system can be consistent and partition tolerant, but not available
- The system can be available and partition tolerant, but not consistent

This feature is particularly useful when a large amount of data needs to be replicated, because the system can be configured to replicate the data in small chunks in the background, making the copying process more efficient. Copying data in small chunks does not stress the hardware as much as copying the whole file at once. Additionally, copy operations are done over LAN or WAN, both of which may fail at some point. Chunks sent over a network may be used to resume the data transfer if a network failure occurs. This way, the system stays more responsive during this process. This is called the eventual consistency model: it is used to describe a distributed system that manages data with some latency factor, making them inconsistent in short periods, but eventually, when the system finishes with its operations, data will be consistent. Those systems are also known as systems that provide BASE (Basically Available, Soft State, Eventual consistency) semantics. Digital libraries are ideal candidates for the deployment of distributed database systems, since digital content is huge and any changes in the content itself are rare. This

enables the system to enforce a disaster recovery scenario using an eventual consistency feature of distributed database systems that can efficiently handle large binary files.

## Software architecture

### Current approach

In order to estimate the benefits of a suggested solution, it is important to analyze the current approach for deployment of disaster recovery scenario and compare it with the proposed one, based on the MongoDB database system. Disaster recovery plan assumes that it is necessary that customer data should reside in multiple locations. Current approach requires two data locations for duplicated customer data: one inside the data center and one outside of the data center. Data copy should be stored inside the data center for protection against the loss of individual data server within a facility. Furthermore, data copy should be stored outside the data center for complete protection against the loss of the data center. These actions are conducted using procedures that perform data copying between locations. One typical tool for performing this scenario on Linux operating systems is Rsync. This solution is good when the number of servers for synchronization is not too big, the amount of data is not too large and the number of files marked for synchronization is not very big. If these requests are not satisfied, problems can occur. Some issues that can occur:

- Synchronization jobs begin to overlap
- Synchronization jobs times increases
- Many simultaneous jobs overload servers or networks
- Problems with coordinating additional steps needed after Rsync job completes
- Problems with monitoring jobs and alerting on failures

### MongoDB GridFS architecture

Unlike RDBMS, MongoDB stores data in BSON documents (Chodorow, 2012). BSON file specification is similar to JSON file specification with added support for several data types. Documents are grouped into collections that correspond to tables in RDBMS and collections are grouped into databases. Databases and collections in MongoDB database system do not enforce schema and there are no relations between collections. Every document in a collection is uniquely identified by _id value, which is a primary key in MongoDB collection. There are two scenarios for managing this value: it can be automatically generated by MongoDB or it can be inserted by the users. If automatically generated by MongoDB, this _id value is similar to a universally unique identifier (UUID) or globally unique identifier (GUID). Each BSON document can hold up to 16 MB of unstructured data, which is not sufficient for storing digital content, because digital content is usually bigger. However, MongoDB includes a specification called GridFS and this specification is developed exclusively for handling large unstructured content, bypassing size limit boundaries. GridFS stores files on the file system, much like the operating system itself. In addition, each file is split into chunks and every chunk is a separate document. The default chunk size is 256 KB, but this value can be adjusted for achieving better

file manipulation performances. The adjustment is usually performed in compliance with the operating or file system. The benefit of splitting files is that GridFS can handle very large files, regardless of the file size and the underlying file system. Besides splitting the file, GridFS also stores additional information about a file that is handled, such as file name, MD 5 hash, upload date and similar metadata in a separate collection. Some important features of the MongoDB and MongoDB GridFS are:

- Automatic replication in MongoDB and MongoDB GridFS can be enabled
- Scalability with minimum administrative effort through sharding can be achieved
- Partition tolerance can be designed in order to endure a needed number of server outages
- It is possible to improve performance of file handling operations because increasing the number of servers included in infrastructure further increases availability and performance of the IS

Apart from these characteristics, some features are specific to MongoDB GridFS:

- File streaming is enabled for GridFS
- GridFS does not limit the number of files for handling
- GridFS does not impose file naming limitations
- Grids stores files into pieces that are 256 kilobytes in size
- Storing files in small pieces enable efficient replication, sharding, streaming and file handling in general
- GridFS stores metadata separately from file pieces and creates two lists for these purposes which allows search operations directly in GridFS architecture, if needed
- GridFS structure is stored in 2 gigabyte pieces, making database size limited only by hardware resources

## Replication

Replication is a feature of MongoDB that requires several instances of MongoDB data servers configured as a replica set. Replication is the process of synchronizing data across multiple servers. It provides data redundancy and high availability, ability to recover from hardware failure and disaster recovery planning. If secondary/replica servers can be used for reading operations, the configuration is suitable for load balancing operation. Single replica set consists of several MongoDB instances that manage the same data sets. One of those instances called primary handles all write requests. Other instances, known as secondaries manage replicated data and they can be used only for read requests. One replica set can have one primary database server and several secondaries. Every member of a replica set must be able to reach all other members and this mechanism is triggered every two seconds. This process is called the heartbeat. If the primary server does not reply within 10 seconds from the last heartbeat, then this primary server is considered unavailable. At this point, other members of the set will start the election process and elect the new primary. The election process is auto initiated because downtime of the system should be minimal. There are a number of factors that affect elections, such as priority, last write time and the number of connections to other members.

A secondary cannot become the primary if:

- The last write time is not recent, i.e. other member of the set is more up-to-date
- It has a lower priority than any other member of the set
- Cannot reach the majority of the set

Priority is a numeric value between 0 and 100, representing eligibility of a member to become a primary. Higher number makes the member more eligible, while priority of 0 means that a member will never become a primary. Setting priority for members is a useful mechanism for giving preference to a particular server for its hardware or network connectivity. Members with priority set to zero still replicate data, accept read operations and vote in elections.
Elections are the process of electing a new primary in a replica set. This process occurs after initiating a replica set and every time the primary becomes unavailable. Each member of the configured replica set has one vote. The first member to receive the majority of votes becomes a new primary. There is a maximum of seven voting members in set and 12 set members in total. This is to reduce the amount of network traffic and to decrease the duration of the elections. During elections, all members of the set can veto an election for a reason, important ones being:

- Member seeking an election has a lower priority than another member
- Member seeking an election is not up-to-date with the most recent write operation

Another important part of the election process is the majority. The majority can be defined as more than half of all members in the set. Table 1 illustrates majority:

**Table 1: Majority in a typical replica set**

| Members in the set | Majority |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 2 |
| 4 | 3 |
| 5 | 3 |
| 6 | 4 |
| 7 | 4 |

Majority, priorities and the election process can influence the performance of the set, so they must be given serious thought when designing a replica set. Good design practice is to have an odd number of set members and to put majority of those in a single data center if possible. That way the set can always elect a primary and loss of network connectivity between data centers will not trigger the elections. If the network between data centers malfunctions, the set would still have majority in the data center 1. Secondary server in the data center 2 would not be able to contact others, but it cannot become a primary. This way only one primary is accepting writes.

Second secondary will replicate the data without inconsistencies as soon as the network connectivity is restored.

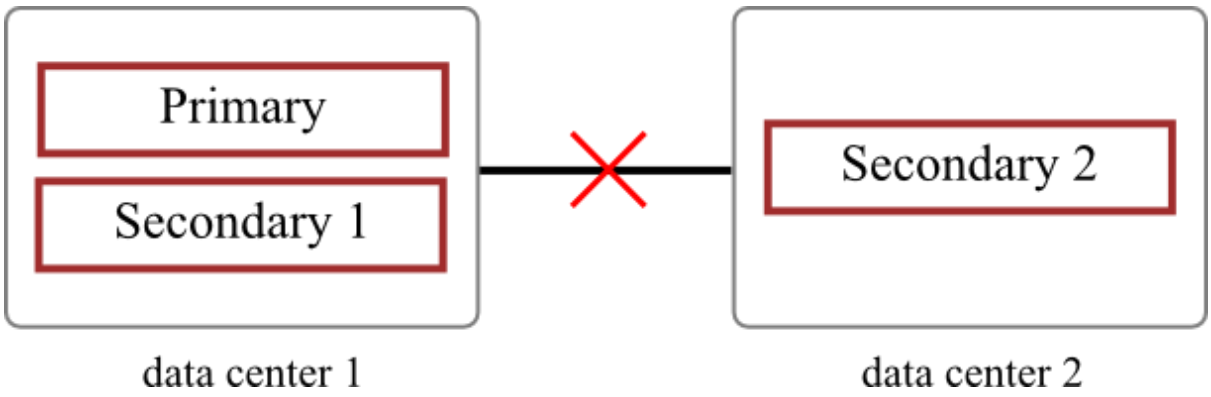


**Fig. 1. Example of a good practice in designing a replica set**

## Sharding

Sharding, or horizontal partitioning, is the process of splitting data across multiple servers. Each server hosts a subset of data, thus decreasing the load on the servers. Reaching the storage capacity does not mean that a more powerful server is needed. Adding another less powerful server will trigger load balancing in the background, enabling the database system to handle more data. This is a way in which even commodity servers can be deployed.

Many RDBMSs support manual data sharding. When performing manual sharding application must keep several connections with several database servers. This approach works well for a small number of servers. However, when adding new servers to the shard or when changing load patterns, the configuration becomes more complicated, therefore making the approach difficult to maintain. Similar problems are present in environments where file system based repositories are selected for handling large digital content. In both cases, the application needs to be altered in order to become aware of new locations for storing digital content.

MongoDB supports auto sharding by default. Client applications are not aware that they are querying multiple servers in a shared environment because auto sharding abstracts the architecture. In addition, MongoDB automatically balances the data across shards, hence facilitating horizontal partitioning.

MongoDB shards the data at the collection level by the user defined shard key. Shard key can be an indexed field or indexed compound field that must exist in every document in the given collection.

## Results

In order to demonstrate the validity of the approach, the experiment was conducted. The replica set was created using three servers in a specific computer network: two servers were stationed inside the data center consisting a local area network. The third one was dislocated on a secure and distant location. The third server was connected to a data center over the internet, and this part of a configuration was a key part of a disaster recovery scenario, as shown in Fig. 2.
Configured like this, the system can survive various server outages, including the loss of a server in the data center, the loss of a server in a distant location or even a loss of the complete data center, which is the real disaster recovery scenario.

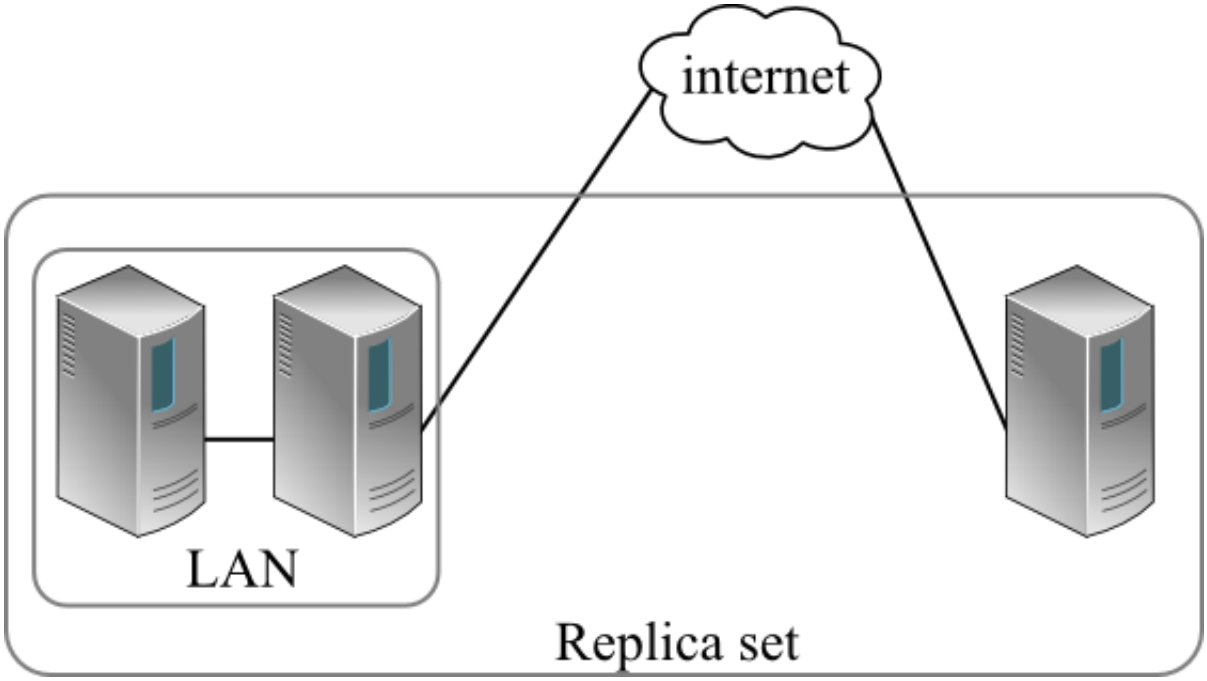


**Fig. 2. Disaster recovery implementation using replica set**

After establishing a replica set, an experiment was conducted and times are shown in the table 2. Characteristic files were inserted into the primary MongoDB server and times needed for local and distant replication were measured. Specific sizes that are expected in a production environment were chosen for the experiment. Following eventual consistency rules, the client will receive a confirmation that a file is uploaded immediately after write operation is completed on the first server.

**Table 2: Replication time in different surroundings and for different file sizes (WAN download speed is limited to 1 Mbps and LAN download speed is limited to 100 Mbps.)**

| Network type | File size | Replication time | Command copy time |
|---|---|---|---|
| Local area network | 5 MB | Less than 1 second | Less than 1 second |
| Wide area network | | 54 seconds | 54 seconds |
| Local area network | 25 MB | 3.2 seconds | 3 seconds |
| Wide area network | | 4 minutes 11 seconds | 4 minutes 11 seconds |
| Local area network | 50 MB | 7 seconds | 6 seconds |
| Wide area network | | 8 minutes 32 seconds | 8 minutes 32 seconds |
| Local area network | 100 MB | 13 seconds | 11 seconds |
| Wide area network | | 17 minutes 36 seconds | 17 minutes 36 seconds |

Measured times present average values from three separate experiments conducted on networks with normal, day to day activity. Server and network failures were simulated by unplugging either the power or network cable during the testing.

## Discussion

The proposed architecture appeared simple and easy to control. The data center is organized into a local area network and a separated distant location was engaged. This location contains data that is identical as data inside the data center. As shown in table 2, times needed for data to become consistent are similar to the times needed for basic operating system command copy to complete. This confirms that time and information overhead during the process of replicating data is minimal.

As the system will handle files that are unlikely to change often, this approach is very useful and achieving disaster recovery is completely automatic. A proposed solution to the problem involves usage of MongoDB database systems and its GridFS specification for handling binary files. Once transferred to the database, files were handled together with the rest of database content, which enabled automatic replication.

Although the results presented in table 2 represent analysis of a small system with only three servers, it is expected that in a bigger system with more servers measured values would not increase. Considering this, the proposed solution is becoming more attractive with the increase of a number of servers in a system, since the time needed for achieving complete consistency does not increase.

## Conclusions

Handling binary files using RDBMS or file system based repositories caused problems with implementing efficient disaster recovery strategy. Therefore, a different approach was needed. The proposed solution uses NoSQL database system, MongoDB in particular, to resolve this issue. The solution relies on GridFS specification, a custom specification embedded into MongoDB for efficient handling binary files. Once inside the database system, files could be treated using features that already exist as integral part of the MongoDB database system. Using the eventual consistency model, a disaster recovery scenario created and tests showed that the solution is appropriate for conducting disaster recovery scenario, regardless of which server would stop responding. Even the complete loss of the data center was tolerable, which is the main goal in conducting disaster recovery.

## Implications for research

Further research could take place in measuring the performances of a system that would rely on different muscle solution for handling binary files. An interesting application in IS could be the deployment of used servers as NoSQL instances. Another interesting topic would be the development of an architecture that consists of both, replica sets and shards. This approach would enable the utilization of used and/or commodity hardware, which can be very important for poor economies in particular.

## Implications for practice

Apart from systems that need a disaster recovery scenario, other IS that handle binary files and similar content can benefit from the use of NoSQL database systems. Implementing solutions that include NoSQL databases for managing digital content include, but are not limited to:

- Using MongoDB database system for handling unstructured data can be deployed in existing information systems
- MongoDB database system is suitable for handling large amount of documents because it incorporates sharding/horizontal partitioning
- Horizontal scaling and distributive characteristics enable the system to survive server termination
- Even old and used hardware could take place in a system like this

## Limitations

The study shows acceptable adoption of NoSQL technology inside particular SOA and any insight is limited to this context. When applied in different surroundings, the model could require adaptation.

## References

Alier, M.F., Guerrero, M.J.C., Gonzales, M.A.C., Penalvo, F.J.G. and Severance, C. (2010), "Interoperability for LMS: The missing piece to become the common place for e-learning innovation", *International Journal of Knowledge and Learning*, Vol. 6 No.2, pp. 130-141.

Beggan, D. (2011), "Disaster recovery considerations for academic institutions", *Disaster Prevention and Management*, Vol. 20 No. 4, pp. 413-422.

Bull, S. and Reid, E. (2004), "Individualized revision material for use on a handheld computer", *Proceedings of mLearn 2003, London, UK, May 19-20, 2003*.

Chodorow, K. (2013), *MongoDB: The Definitive Guide, 2nd edition*, O'Reilly Media.

Chang, Y., Wilkinson, S., Potangaroa, R. and Seville, E. (2012), "Managing resources in disaster recovery projects", *Engineering, Construction and Architectural Management*, Vol. 19 No. 5, pp. 557-580.

Casany, M.J., Alier, M., Mayol, E., Piguillem, J., Galanis, N., Garcia-Penalvo, F.J. and Conde, M.A. (2012), "Moodbile: A Framework to Integrate m-Learning Applications with the LMS", *Journal of Research and Practice in Information Technology*, Vol. 44 No.2, pp. 129-149.

Guster, D., Lee, O. and McCann, B. (2012), "Outsourcing and replication considerations in disaster recovery planning", *Disaster Prevention and Management*, Vol. 21 No. 2, pp. 172-183.

Kaur, T. (2009), "Disaster planning in university libraries in India: a neglected area", *New Library World*, Vol. 110 No.3/4, pp. 175-187.

Kusumasari, B. and Alam, Q. (2012), "Local wisdom-based disaster recovery model in Indonesia", *Disaster Prevention and Management*, Vol. 21 No. 3, pp. 351-369.

Kossmann, D., Kraska, T. and Loesing, S. (2010). "An Evaluation of Alternative Architectures for Transaction Processing in the Cloud", *ACM International Conference on Management of Data*, pp. 579–590.

Naismith, L., Lonsdale, P., Vavoula, G. and Sharples, M. (2004), "Literature review in mobile technologies and learning", *FutureLab: Innovation in Education*, Scientific Report, available at: http://archive.futurelab.org.uk/resources/publications-reports-articles/literature-reviews/Literature-Review203 (accessed 4 October 2013).

Sharples, M., Taylor, J. and Vavoula, G. (2005), "Towards a theory of mobile learning", *Proceedings of mLearn 2005, Cape Town, South Africa, October 25-28, 2005*, pp. 1-9.

Stonebraker, M., Madden, S., Abadi, D.J., Harizopoulos, S., Hachem, N. and Helland, P. (2007), "The End of an Architectural Era (It's Time for a Complete Rewrite)", *VLDB '07 Proceedings of the 33rd international conference on Very large data bases*, pp. 1150-1160.

Vavoula, G.N. and Sharples, M. (2002), "KLeOS: A personal, mobile, knowledge and learning organization system", *Proceedings of the IEEE International Workshop on Mobile and*

*Wireless Technologies in Education (WMTE2002) Vaxjo, Sweden, August 29-30, 2002*, pp. 152-156.

## Biographies

**Milorad P. Stević** is a Database Architect with 15 years of experience (Progress versions 8,9; Oracle version 10g, MSSQL Server versions 2005, 2008, 2012), He is also a .NET Software Architect with a 8 years of experience in development for the Windows platform, Web, Web Services, Client/Server and n-tiered distributed applications, broad experience in all parts of project life cycle including the analysis, design, development, implementation testing, debugging/profiling and support. He is a full-time faculty member at The Higher Education Technical School of Professional Studies, Novi Sad, Serbia.

**Ivica Kolenkaš** is currently a student at the Higher Technical School of Professional Studies, Novi Sad, Serbia, on the Department of Information Technology. His interest is focused on open source technologies as Linux administration, programming in Python and advocating the open source philosophy. He has authored papers at international conferences.