RainbowFS:
Modular Consistency and Co-designed Massive File system

Cohérence modulaire et conception conjointe d'un système de fichiers massif

Appel à projets générique ANR 2016
PRCE (Projet de recherche collaborative — Entreprises)

Défi 7. Société de l'information et de la communication
Axe 7 : Infrastructures de communication, de traitement et de stockage
Axe 3 : Sciences et technologies logicielles
Axe 5 : Données, Connaissances, Données massives (Big Data)

Orientations de la SNR concernant le Défi 7 :
26 : 5è génération des infrastructures réseau (déploiement à grande échelle de l'internet des objets) ;
27 : objets connectés (architectures logicielles distribuées) ;
28 : Exploitation des grandes masses de données (stockage).

12 April 2016

Abstract

RainbowFS proposes a "just-right" approach to storage and consistency, for developing distributed, cloud-scale applications. Existing approaches shoehorn the application design to some predefined consistency model, but no single model is appropriate for all uses. Instead, we propose tools to co-design the application and its consistency protocol. Our approach reconciles the conflicting requirements of availability vs. safety: common-case operations are designed to be asynchronous; synchronisation is used only when strictly necessary to satisfy the application's integrity invariants. Furthermore, we deconstruct classical consistency models into orthogonal primitives that the developer can compose efficiently, and provide a number of tools for quick, efficient and correct cloud-scale deployment and execution. Using this methodology, we will develop an entreprise-grade, highly-scalable file system, exploring the rainbow of possible semantics, which we will demonstrate in a massive experiment.

Keywords: Cloud Computing; Consistency; File Systems; Storage; Geo-Replication; Static Verification; Distributed Application Software Engineering; Big Data.

CONTENTS

I Administrative information / Informations administratives

I.1 Table of people involved in the project / Tableau récapitulatif des personnes impliquées dans le projet

Table 1 summarises the people involved in the project.

Table 2 shows how the manpower is distributed, first by partner and task, second by partner and type.

I.2 Changes from the pre-proposal / Évolutions éventuelles par rapport à la préproposition

The major change with respect to the first-phase summary is the requested duration of the funding period. ANR plans to announce the funding decision in July 2016. As this is a very unfavourable period for recruiting the best PhD students, we are requesting a 48-month duration of the funding period to cope with uncertainty. The workplan itself takes 36 months, with M 01 starting when the project students and staff is hired.

The budget has increased slightly, because our initial estimate ignored the institutional overheads.

II Context, positioning and objectives of the detailed proposal / Contexte, positionnement et objectif de la proposition détaillée

II.1 Objectives and challenges

RainbowFS proposes a "Just-Right Consistency" approach to developing geo-distributed applications, whereby an application pays only the consistency price that it strictly requires. The tools contributed by the project will *prove* that the application is safe, while at the same time ensuring that it is *highly available and scalable*. Other tools will contribute to the *rapid deployment* and to the *monitoring and analysis* of system behaviour and performance. The project applies this approach to the design of a high performance, highly scalable, tunable *geo-replicated file system*.

Current cloud storage offerings are divided between opposing, monolithic, one-size-fits-all consistency models. The RainbowFS partners believe that this thinking is obsolete. We turn the monolithic, predefined model on its head, and propose a methodology by which the system is *as weakly consistent as possible, yet sufficiently consistent to be safe*. Thus, the application pays the consistency price it strictly requires, and no more. We call this approach "Just-Right Consistency."

Our **first challenge** towards this vision is to guide the developer, and help her make correct choices in the safety-vs.-availability trade-off. Availability is an essential enabler for performance and scalability, since, to be always available, a replica must avoid synchronising with others [1, 16]. We aim to develop a *co-design methodology that reconciles correct application development with high availability*. The co-design approach is enabled by a logic that was previously developed by the academic partners [12, 31, 54–56]. The logic enables verifying formally that the application's *integrity invariants* are satisfied by a given (weak) consistency protocol. If not, a tool implementing the logic will provide a counter-example, which helps the developer diagnose and fix the problem. Then, to restore correctness, the developer has two choices: either to weaken the specification, at the risk of anomalous behaviour, or to strengthen the synchronisation, decreasing availability. Which direction to take is a design decision, but in either case, the tool verifies that the fix actually restores correctness. In this way, the developer co-designs the application specification and the associated consistency protocol. Our approach avoids minimises synchronisation, thus enabling availability, to what is strictly needed, while proving that the application is nonetheless correct.

Our **second challenge** is to translate the resulting specification into a running, efficient, deployable application. We deconstruct consistency models into *composable primitives* along three dimensions that structure the consistency design space. Correspondingly, we will develop a library of components that the application developer can combine into a functional application skeleton, paying only for what is strictly needed. These components are bound together by a *composition framework* that ensures that the composition is highly efficient, for instance by consolidating together any redundant messages or waits. Finally, this task provides a set of deployment, monitoring, analysis and debugging tools to aid the programmer achieve the

Partenaire	Prénom Nom	Emploi	Implication sur la durée du projet (personnes.mois)	Rôle et responsabilités dans le projet
Inria Paris (Regal)	Marc Shapiro	Dir. de recherche	18	PI (Coordinateur scientifique) Replication and consistency at large scale.
	Sébastien Monnet	Maître de Conf.	9	Replication, consistency and data placement
	Mesaac Makpangou	Chargé de rech.	9	Modular consistency, protocol synthesis
	— TBA	Doctorant	36	Modular consistency, application co-design
	— TBA	Post-doc	36	Modular consistency, project coordination
Scality	Vianney Rancurel	Director of research	6	Industrial-grade object and file storage
	— TBA	SW Engineer	36	File system, experiments, integrate with Scality products
CNRS-LIG	Vivien Quéma	Professeur	9	Modular consistency, fault-tolerance, coordination of large-scale Cloud experiments
	Renaud Lachaize	Maître de Conf.	15	Tools for efficient and correct modular execution, coordination of large-scale experiments
	— TBA	Doctorant	36	Efficient component library for modular consistency
Télécom SudParis (TSP)	Pierre Sutra	Maître de Conf.	18	Large scale replication and file system
	— TBA	Doctorant	36	File system

Table 1: People involved in RainbowFS

	Inria Regal	**Scality**	**CNRS-LIG**	**TSP**	*Total*
Task 1	45	8	10	6	*69*
Task 2	27	8	40	18	*93*
Task 3	36	26	10	30	*102*
Total	*108*	*42*	*60*	*54*	*264*
Permanent staff	36	42	24	18	*120*
PhD students, interns	36	0	36	36	*108*
Post-docs	36	0	0	0	*36*
Total	*108*	*42*	*60*	*54*	*264*
Taux de précarité					*14%*

Taux de précarité is the ratio of non-permanent personnel, excluding PhD students and interns, to the total.

Table 2: Distribution of manpower per partner: by task, by type

dual goals of safety and availability.

The **third challenge** is to apply our approach to a real, useful and challenging application, and to validate experimentally that our goals have been met. Given the market demand for structured storage (discussed later in this document), we target the development of an *entreprise-grade, highly available, performant and scalable file system*. The file system will be co-designed and proved correct according to the Just-Right Consistency approach. It will be implemented and deployed using the tools of modular consistency.

Because client needs vary, RainbowFS will study the *full rainbow of reasonable storage semantics*. At this level too, users should pay only for what they need. Applications will be able to count on a Posix-like naming tree and powerful synchronous commands, such as `rename`. An application that manages a flat hierarchy of unique, write-once files, should enjoy similar performance to an object store, but remains confident (thanks to the tools of Task 1) that the application remains correct under these weaker guarantees. An application with extreme latency requirements (e.g., an interactive game) should be able to request replicas very close to the end-user; this requires the file system to hundreds or thousands of partial replicas. The scale and correctness of this file system will be validated in a *massive-scale geo-distributed experiment*.

II.2 Context

Cloud computing and innovation. Cloud computing is motivated by the potential of economies of scale, resource consolidation, access to unlimited resources, and service automation. The cloud market has been so far captured by a few oligopolies, each running a small number of vast data centres, essentially based in the USA. The appeal of the cloud is likely to continue and to capture an increasing fraction of demand for computing.

However, current cloud architectures do not satisfy emerging client requirements for millisecond response, security and privacy, and integration of private resources into the cloud. Especially in Europe, communities and governments provide local resources, and 5G networks will deploy mini-datacentres at the edge. Social networks, open data, electronic commerce, scientific experiments, the Internet of Things will continue to feed an explosion of data sources and needs for ever-greater amounts of computation and storage.

According to this so-called "Fog Computing" vision, cloud resources will grow in number and variety. Future clouds will collect high numbers of diverse and geographically distributed data centres. Future cloud stores will consist of hundreds or even thousands of geo-distributed sites [17].

The cloud is an area of rapid innovation, calling for a close cooperation between academic research and entrepreneurship. It raises deep research questions, for instance in the area of distributed computing, which call for concrete solutions that pass the test of reality. The French innovation ecosystem is well positioned, with very active and successful SMEs, feeding on a rich soil of academic research in distributed computing and related areas.

The RainbowFS proposal grows out from previous collaboration between an SME and academia. It aims to address the difficulty of developing applications that are both correct and make good use (from both an economic and performance perspective) of large-scale distributed resources at Fog-computing scale.

Replication and consistency models. Today, programming a distributed application remains something of a black art. Programming requires to be able to predict what the machine will be doing, but a distributed system has many independent moving parts, and is subject to unpredictable additions and removals of resources, due to upgrades, failures, or flash load variations. Furthermore, the requirements themselves are often ill-defined and change over time. This requires a fast development path that ensures that the application requirements are met, within a changing execution environment.

For performance and fault-tolerance reasons, inevitably, the same information is duplicated. For instance, a given data item may have many redundant copies, e.g., across a RAID, backed-up on tape, cached in memory, and replicated in multiple data centres Furthermore, different data items are logically related: for instance, accounting rules require that the balance of all accounts be a constant, therefore, increasing one account requires decreasing another.

This raises the issue of *consistency*. Any update needs to be validated against *integrity rules* (such as the constant balance rule above) and *propagated* to all copies of the data item and to the related data items. This is not instantaneous; measured intercontinental round-trip times are in the order of hundreds of milliseconds (France-US East Coast: 90–120 ms; France-New Zealand 300–600 ms). Therefore, perfect consistency is not practical, and an application runs above a less-than-perfect *consistency model* provided by the system.

A so-called *strong model*, one that performs updates in a total sequential order is easy to understand for developers. However, when network failure occurs

(which is inevitable), the strongly-consistent model blocks rather than risk returning a wrong answer, and everything grinds to a halt. Thus, the strong model inherently sacrifices availability (and thus performance and scalability) to maintain a high standard of consistency. The cost and difficulty of strong consistency increases with latency and number of sites; in a Fog Computing set-up with thousands of sites, is is just not possible.

Alternatively the system can remain *available*, i.e., continue to accept updates, at the risk of returning anomalous results [30]. Such a *weaker* consistency model admits more parallelism and enjoys better performance, but exposes the application to anomalies, which confuse the programmer and may lead to data loss or violation of integrity.

The performance difference is substantial. In controlled benchmarks, we measured a 13.5× performance improvement (3× lower latency and 4.5× better throughput) between the strongest and the weakest model in a small-size cloud [6]. These numbers mean that synchronisation wastes enormous amounts of money and energy costs due to synchronisation. Projecting the above numbers, under weak consistency, the same workload might cost up to 13.5 times less, in hardware and energy resources, than under strong consistency.

On the other hand, because of the need to cope with anomalies at the application level, the development cost is much higher.

This inherent trade-off between safety on the one hand, and availability on the other has sparked the development of many competing *consistency models*, approaches, and acronyms: SQL, NoSQL, NewSQL, AP, CP, KVS, object storage, file systems, databases, etc. Application developers must make a bet in advance of which model to chose, that will be most appropriate to their needs. Some systems provide strength parameters [10, 13, 15, 48, 73, 81], but all lack tools and guidance to ensure that their application will behave as intended over the model provided.

Storage systems. The need for data storage is expanding at a high rate. User-generated data, such as social networks and videos, is already radically pushing requirements, to the order of petabytes. This is dwarfed by the volume to be generated by IoT devices, or by scientific experiments such as LHC at CERN. As a result, there is demand for efficient, dependable, geo-aware and low-cost storage systems.

Increasingly, storage is *software defined*, i.e., is built as a logical network of commodity components. Data is *sharded and replicated* within data-centres for parallelism fault tolerance, and *geo-replicated* for availability and access latency. In fact, a cloud-scale storage system such as Amazon S3, Cassandra or HDFS often forms the bottom layer of internet-scale services.

Data storage systems are currently divided, based on how they address replication and consistency. Before even starting to store their internet-scale data and to develop applications to exploit it, developers are forced to choose upfront between antagonistic solutions.

Traditional file systems are a hierarchy of folders and files. Files are updated in place, i.e., an update overwrites the previous state. File systems are often considered slow and non-scalable. Indeed, because of its complex transactional semantics (e.g., `rename`), existing systems all require strong consistency to some degree. File systems are here to stay, because legacy applications (e.g., databases, custom software, or CAD software) rely on their structure, and because their hierarchy provides an intuitive way to organise and to secure information.

Nonetheless, for reasons of scalability and performance, a current trend is towards low-level *object storage* systems, similarly to the NoSQL trend in databases. This approach forgoes the file system tree in favour of *buckets*, which are similar to very large, flat folders. Furthermore, each update of an object creates a new *version*. In this way, complex synchronisation is avoided in the common case. These are widely used in modern applications (e.g., Web 2.0 and edge networks).

II.3 Project outcomes, results, and products

This project advances both the principles and the practice of distributed computing in general, and of Fog-scale storage in particular.

RainbowFS will expand and apply the new Just-Right Consistency approach to the design of distributed computing and storage systems. This turns the traditional model-oriented approach on its head. So far, consistency models have been designed in a generic, application-independent way. This has been productive in better understanding the design space. However, no single consistency model is uniformly better than the other, due to the contradictory imperatives of safety and availability. Shoehorning an application to conform to a model that was predefined in a vacuum, is not a productive use of developers' skills. Just-Right Consistency instead is an approach that co-designs the application and its consistency protocol, thus enabling the developer to semi-automatically identify the

strict minimum of synchronisation that her application needs.

To support the Just-Right Consistency approach, the RainbowFS project will develop tools to precisely identify the problematic concurrency patterns, to help with rapid development and deployment, and to monitor, analyse and understand the behaviour of applications. One such tool is a static analysis engine, based on the CISE logic [12, 31, 54–56], which proves, with polynomial complexity, whether a given application running above a given consistency protocol maintains a given invariant. If not, other tools will leverage the counter-examples returned by the CISE tool to precisely identify the problem and to suggest solutions, such as a weakening of the application specification, or a strengthening of synchronisation. The static tools will be complemented by dynamic ones, for deploying and controlling the distributed application, while monitoring its performance and correctness. Another tool is a library of consistency primitives that can be combined efficiently, leveraging ongoing research by the partners that organises the consistency design space into three logical dimensions.

Using the tools described above, the project will tackle a real and demanding use-case. We will analyse, design, prove correct, build, deploy and experiment with a new distributed file system. The file system has several important objectives: *(i)* to support the extreme data sizes (hundreds of petabytes) required by modern big-data applications, *(ii)* to support the thousands of sites, of heterogeneous size and computing power, foreseen for future Fog-computing networks, *(iii)* to converge the file-system and object-storage approaches, and *(iv)* to support a Just-Right Consistency approach for the file system's users themselves, allowing them to safely parameterise its semantics to minimise cost while guaranteeing higher-level correctness. The industrial partner, Scality SA, plans to integrate this file system into its commercial offerings within the timeframe of the project.

Many companies are reluctant to make the move to object storage, because a many existing applications rely on the Posix file system interface. Our design layers the file system above Scality's object store. As we discuss later, a file system actually requires synchronisation only in very limited circumstances, and therefore our design can reconcile the performance of object storage with the safety requirements of legacy applications. Our system will offer the full rainbow of file system semantics, from linearisable to fully asynchronous; applications that require the availability of object stores can have it, and those that require highly-structured file system trees with atomic `rename` can have it too.

Some systems, such as HDFS, take the approach of force-fitting the file system into an eventual-consistency model that breaks the Posix requirements. The originality of our approach is to start from the requirements of the application that runs on top of the file system, and to tailor the consistency models accordingly; hence the "Rainbow" model.

II.4 Related work and partners' previous contributions

The design of efficient and robust large scale distributed storage systems is a very active R&D field, at the core of the cloud revolution. As explained earlier, the market is split between strongly- and weakly-consistent system designs. We hope to help heal this split with our Just-Right Consistency and modular consistency approaches, and to reconcile them in our file system application.

II.4.1 Strong consistency Strong consistency provides familiar guarantees and decreases the effort to make applications correct, but entails frequent synchronisation; each consensus requires two round-trips [47], i.e., several hundred milliseconds in a wide-area network. Strong consistency comes at a high hardware cost; for instance, the very strong guarantees of Google's Spanner require specific hardware clocks and high-quality dedicated fibre network links [21]. Recent research has aimed to decrease the cost of strong consistency, in several ways, such as enforcing total order writes but weakening guarantees on reads [14, 73], or increased internal parallelism through sharding, disjoint-access parallelism, and genuine partial replication [26, 63, 72]. Few practical systems enforce the strongest consistency; even commercial centralised databases do not ensure serialisability by default [9]; the resulting anomalies are considered acceptable with respect to the improved performance. However, the Jepsen experiments [41] highlighted that many production systems suffer from ill-defined guarantees and/or incorrect implementation thereof.

The partners have contributed efficient strong consistency designs. Sutra and Shapiro [77] designed a latency-optimal consensus protocol that parallelises commuting updates. Saeida Ardekani et al. [62] improved snapshot isolation [73] to support genuine partial replication and to improve internal parallelism. Partners designed a throughput-optimal uniform total order broadcast protocol for state machine replication within a cluster of machines [34]. Aublin et al. [8] designed protocols for efficient Byzantine fault-tolerant

state machine replication, as well as a framework to compose such protocols, in order to efficiently support various operating conditions.

II.4.2 Availability and weak consistency To be available despite network partitions, the system must accept concurrent updates without remote synchronisation. This enables good performance, thanks to immediate local response, spreading load in parallel over available resources, and a greater range of implementation choices.

In return, these systems can promise only weak consistency [92]. Such models exhibit anomalies, which require extra effort by the application programmer to ensure safety, increasing the development cost. Example commercial systems include Cassandra [83], Amazon S3 [15], Dynamo [25] or Riak [43].

The partners of this proposal have been strong contributors to research on available storage systems. We have designed and implemented systems that guarantee Causal Consistency and highly-available transactions [9]. Zawirski et al. [93] implemented SwiftCloud, a highly-available, widely-geo-replicated cloud database designed for Fog environments. Akkoorath et al. [4] designed and implemented Antidote, a geo-replicated cloud database designed for high internal parallelism and modularity. Previously, Shapiro et al. [68, 69] invented CRDTs, high-level data types that provide convergence and correctness guarantees to applications above a causally-consistent system. Scality and Inria together designed and implemented a geo-replicated file system featuring CRDT semantics [80].

II.4.3 Hybrid consistency Some designs allow the user to specify a different level of consistency and/or availability on a case-by-case basis [45, 73, 90]. Commercial examples include Cassandra or Riak. Red-Blue Consistency classifies each operation as either blue, commutative with all others and executed asynchronously, or red, non-commuting with and synchronised with other red operations. At a higher level of abstraction, the Pileus key-value store [81] supports flexible and dynamically adjustable consistency-based service level agreements.

However, more consistency options only complicate the developer's task, in the absence of tools to guide the developer and to ensure that the result is correct. Previous static analysis tools include Bloom [5], which identifies points of non-commutativity, Homeostasis [61] which considers conflicts between assignments, and Sieve [49], which helps automate Red-Blue consistency. All three have limited scope and lack theoretical grounding.

The tools previously contributed by the project partners constitute a breakthrough in this respect. Our first major result was Explicit Consistency [10], later improved with the CISE logic [31] and the associated tool [55, 56]. CISE boils down to three correctness conditions: *(i)* Effector safety verifies that every update in isolation maintains the invariant, *(ii)* effector commutativity verifies that concurrent updates commute, and *(iii)* precondition stability checks whether the precondition of one update (required by the first rule) is stable under concurrent updates. The CISE logic supports general properties expressable in first-order logic and was shown to be sound. It is able to prove, with polynomial complexity, that a given application maintains a given invariant above a given consistency protocol. Conversely, it enables to tailor a consistency protocol, such that has enough synchronisation to ensure the application's safety, but no more.

Task 1 aims to build upon and improve the state of the art of hybrid consistency; the other two tasks will build upon this.

II.4.4 Modular consistency The distributed algorithms community has formalised the different consistency models, both for transactional [2] and non-transactional systems [91]. However, previous work has classified the models informally along a single strong-to-weak axis. This linear view is clearly inadequate, since, for instance, Snapshot Isolation and Serialisability are incomparable. Similarly, although two papers claim that causal consistency is "the strongest possible" model compatible with availability in the linear classification [7, 50], both Bailis et al. [9] and ourselves [4, 93] have strengthened causal consistency with transactional guarantees.

The RainbowFS partners have previously contributed to improving the conceptual and practical toolkit of application designers with a modular decomposition of consistency. Aublin et al. [8] show how to decompose Byzantine fault-tolerant state machine replication into modular protocols. Similarly, Ardekani et al. [6] decompose transactional protocols into composable primitives. In this proposal, we plan to scientifically deconstruct consistency models along three (mostly-)orthogonal dimensions: guarantees related to a total order of writes, those related to the causal order of reads, and those on the transactional composition of multiple operations. A model may be stronger than another on one dimension and weaker on another. We believe that this new classification scheme is both scientifically sound and has good explicative value. Task 2 will leverage this decomposition.

II.4.5 File systems File systems have been the subject of much research and implementation. Relaxing Posix semantics, AFS [36] introduces caching and close-to-open semantics. Lustre, GoogleFS and HDFS [29, 64, 70, 82] improve parallelism by storing data separately from metadata; however they rely on a central metadata server. GoogleFS and HDFS improve reliability by storing data blocks in a quorum of servers.

The above systems are designed for a single datacenter. Systems designed for geo-scale, such as GlusterFS [24] XtreemFS [37], do not deliver convincing performance. Google has announced the design of a planet-scale file system, Collossus [21], but technical information is lacking.

A recent promising geo-distributed approach is CalvinFS [84], layered above a key-value store [27]. CalvinFS splits the metadata server into shards; each shard uses Paxos consensus [46] to synchronise its operations. A file operation is implemented as a transaction across one or more logical shards. CalvinFS outperforms HDFS in both volume and I/O performance.

Commercial cloud-based file-systems designed for collaboration, such as Google Drive and Microsoft OneDrive, are limited to a small number of concurrent users. Furthermore, they are unsafe: for instance, in the presence of concurrent `rename` operations, they can lose data [57, 80].

The project partners are active in the area of distributed file systems. Scality is one of the major providers of high-volume, high-throughput, high-distribution software-defined storage. Scality and Inria Regal together have designed and implemented a highly-available file system, geoFS [65, 80]. Currently, they are working together on proving its specification correct and on measuring its performance. Télécom SudParis have proposed FlexiFS [88], a framework to compare file system design and consistency choices.

In RainbowFS, Task 3 aims at designing an efficient and correct file system for Fog-scale computing. To this end, it will apply the co-design approach developed in Task 1, and will rely on the geo-replication modules created in Task 2.

II.5 Relation with ANR and European work programmes

Within the ANR programme's Challenge 7 "Information and Communication Society" *(Société de l'information et de la communication)*, this proposal fits into Axis 7 "Communication, computation and storage infrastructures" *(Infrastructures de communication, de traitement et de stockage)*.

As we propose to verify formally that applications correctly enforce their invariants, this eases the production of safely operating software; thus, our proposal relates to Axis 3 "Software science and technology" *(Sciences et technologies logicielles)*. Finally, RainbowFS is also linked to Axis 5 "Big Data" *(Données, Connaissances, Données massives)*, as it designed for the petabyte scale.

This proposal builds upon the results of previous collaborations. ANR project ConcoRDanT, "Consistency without concurrency control, in Cloud and P2P systems" (ANR Blanc 2010–2014) was the first to study the concept of CRDTs (Conflict-free Replicated Data Types). CRDTs encapsulate replication and concurrency resolution, and ensure convergence by construction [67–69]. CRDTs are a first stab at ensuring the correctness of applications running above asynchronous replication, and are one of the building blocks of the RainbowFS proposal. In particular, the RainbowFS file system will be developed in Task 3 using CRDT data types.

European FP7 project SyncFree (2013–2016) is exploring large-scale computation without (or with minimal) synchronisation. SyncFree supports a highly-available cloud database that scales to extreme numbers of widely geo-distributed replicas, yet ensures the safety of applications executing at the edge. Extreme-scale distributed and consistency protocols, the insight of strengthening causal consistency, and its formalisation with the CISE analysis, to be carried out in Task 1, are outcomes of SyncFree.

Finally, the main ideas underlying the massively-distributed file system design, to be studied in depth and implemented in Task 3, are the outcome of previous joint work (a CIFRE industrial PhD thesis) between Scality and Inria Regal.

II.6 Feasibility, risks and risk mitigation

This project is ambitious but feasible. In fact, the partners have already proof-of-concept designs or prototypes for several of the contributions targeted in this project.

Regarding Task 1, the CISE logic and its proof of soundness have been published at POPL, the premier venue in language and verification [31]. A prototype CISE analysis tool has been developed [55, 56] and demonstrated on some proof-of-concept examples, including a simplified model of the file system [54]. However, it is only a restricted research prototype, which is too complex and low-level for general use.

Similarly, the partners have experience towards the studies of Task 2. They have published preliminary

deconstructions of consensus protocols [8] and of transactional protocols [6]. They have accumulated experience in tools for deployment and monitoring distributed systems [59, 60].

The industrial and academic partners are very experienced in cloud-scale storage systems and consistency [3, 10–12, 28, 35, 40, 62, 77, 89, 93]. The work proposed for Task 3 builds upon a previous collaboration between Scality and one of the academic partners to design a geo-scale, highly-available file system [65, 80].

We describe some risks specific to the tasks of the project in the corresponding sections (see Section III.1). Furthermore, we identify some overall risks for the project:

1. Unfavourable timing of the announcement of the funding decision (planned for July 2016). At this time of the year, the best PhD students have typically already committed to a project. To manage this risk, we request a project duration of 48 months, which will give us extra hiring flexibility. Within those 48 months, the workplan takes 36 months, numbered M 01 to M 36, starting from when the staff is hired.
2. Failure to develop programs that reconcile availability and safety. This may be, either because application semantics turns out to be too difficult to specify, or because any useful semantics requires a lot of synchronisation. Given recent advances in the state of the art, we think this is unlikely [12, 31, 48, 49, 55, 56, 61]. To manage this risk, we can always use a stronger consistency model, at the expense of availability.
3. Given the many difficulties of developing, configuring and deploying a distributed system, we might fail to scale to the targeted metrics. We mitigate this risk by using state-of-the-art development and deployment technology, and by developing further tools for deployment, monitoring, debugging and analysis in our Task 2.

III Scientific and technological programme, project organisation / Programme scientifique et technique, organisation du projet

The aim of RainbowFS is to develop and validate a novel approach to the design of distributed applications.

Today, the developer must start by choosing a specific consistency model, and bet in advance that it will provide the safety-vs.-availability tradeoff that is right for her application. Although, as explained earlier, some platforms support selecting different guarantees at runtime, they do not help to ensure that the result is correct. In the application area, a number of distributed file systems exist, but they are monolithic, and are not designed for geo-replicated operation at large scale.

In contrast, RainbowFS aims for Just-Right Consistency, adjusting the application semantics and the consistency protocol together. As a result the system is as available as possible, yet sufficiently consistent to be safe; the application pays only the price that it strictly needs. This approach is supported by static and dynamic tools that verify that the resulting application is correct and efficient. It is served by a library and API that deconstructs classical consistency models into independent primitives, from which the developer can combine the minimal set that is necessary. A composition framework ensures that the combination remains highly efficient. This comes with deployment, monitoring, analysis and debugging tools. The approach will be used in practice on a demanding and useful application: an entreprise-grade file system, tunable according to the needs of its users, designed to be deployed on large numbers of geo-distributed sites. It will be validated in a large-scale experiment, to be deployed across several continents.

III.1 Decomposition into tasks

Table 3 summarises the tasks, their timing and dependencies, and the milestones of the project.

Table 4 lists all the deliverables.

We count the first month of the project, designated M 01, when the PhD students are hired. The project terminates 36 months thereafter at M 36. However we request ANR funding duration of 48 months in order to have the flexibility to hire the best PhD students (see Risks, Section II.6).

Recalling Section II.1, the challenges addressed by the RainbowFS proposal are:

1. Design-time tools and guidance to the developer in the co-design of the application and its associated consistency protocol. This challenge is addressed by Task 1.
2. Transforming an application specification, verified in the previous task, into an efficient application skeleton, leveraging composable consistency prim-

	Task 1: Application co-design	Task 2: Modular geo-replication	Task 3: Geo-distributed massive file system
M01–M12	User-friendly, robust CISE tool. Domain-Specific Language to describe operations and invariants. Consider extracting information from source.	Basic consistency component library. Decompose to fine-grain modularity, along three dimensions. Deployment, profiling and monitoring tools.	Single-DC prototype of file system.
MS1	Basic co-design and tools		
M13–M24	Programming patterns. Generate advice to developer for fixing conflicts. Heuristics deriving high-level information from counter-example.	Efficient component library. Composition framework. Analysis and validation tools.	Initial evaluation of single-DC prototype. Geo-distributed file system prototype.
MS2	Functional prototypes		
M25–M36	Synchronisation protocol generation, taking into account workload and faults. Co-design and proof of file system and other applications.	Apply component library to file system and other applications. Large-scale evaluation of library. Running large-scale experiments, using our deployment and monitoring tools.	Large-scale evaluation evaluation of file system, based on real workloads.
MS3	Large-scale evaluation		

Table 3: Table of tasks

Month	Task	Title
M12	1	User-friendly co-design tool
	2	Consistency components; deployment, profiling & monitoring tools
	3	Single-DC file system prototype
M24	1	Synchronisation protocol generator
	2	Efficient composition; analysis & validation tools.
	3	Geo-distributed massive file system prototype
M36	1	Co-design and proof of complete file system and other applications.
	2	Large-scale experimental validation of the component library & tools
	3	Large-scale evaluation report

Table 4: Table of deliverables

itives and tools. This challenge is addressed in Task 2.
3. Experimental and market validation by applying this to a challenging, demanding application. This is the work of Task 3.

The following paragraphs describe each task in detail.

Task 1 : Application co-design

This task is summarised in Table 5.

Objectives of the task: *To ensure that an application is correct without incurring more synchronisation than necessary, thus maximising availability, and enabling scalability and performance.*

Instead of imposing a pre-defined consistency model, we propose to co-design the application functionality and its consistency protocol. The basic insight is that, even within an application, consistency requirements vary depending on the specific operation [48]. Some operations can safely run asynchronously in parallel, while others require synchronisation. A recent breakthrough is the CISE static analysis [12, 31, 54–56], which verifies statically (in polynomial time) whether a program has sufficient synchronisation.

The challenge of this task is to provide design-time and run-time tools to guide the developer in the co-design of the application and its associated consistency protocol. The tools shall verify that the combination is safe, i.e., updates have the intended semantics and verify the system's integrity invariants. When the tools detect an anomaly, they should provide at least a counter-example, and better still, suggest ways to remove the anomaly.

Our co-design approach uses (low latency, available) asynchronous operations by default, ensuring convergence by using Conflict-Free Replicated Data Types (CRDTs) [68]. Using a tool that automatically discharges the CISE proof obligations (the "CISE Tool"), the developer verifies whether the application satisfies its invariants under these conditions. If not, the tool returns a counter-example, which serves as a guide to either weaken the invariants (possibly at the cost of unfamiliar semantics) or to strengthen the synchronisation (at the cost of reduced availability). The analysis-design cycle repeats until verification succeeds. Thus, the developer co-designs the application and the protocol, minimising synchronisation to what is strictly required by its invariants, with the assurance that the result is correct [10].

For instance, consider the invariant that a file system forms a tree. The CISE analysis tells us that concurrent `rename` operations may violate the tree invariant. Therefore, the developer must either change the semantics of `rename`,[1] or ensure that `rename` operations are mutually synchronised. With these changes, the CISE analysis succeeds, thus proving that the file system maintains the tree invariant. The CISE analysis also tells us that the other file system operations do not inherently require synchronisation and that merging semantics is possibe [80].

Methods and technical decisions

Application co-design. We have applied the above methodology successfully to some small examples, including a simplified model of the file system. Ultimately, we wish to achieve the co-design and complete correctness proof of the widely-replicated file system of Task 3. We also plan to co-design other applications, including some that make use of the file system itself.

The specification will model, initially, the Posix file system API [38]. From the invariant that the file system structure must be shaped as a tree, it will derive, for instance, the sequential precondition to `rename`, which forbids to move a directory underneath itself. Furthermore, the Posix specification disallows many concurrent updates, e.g., writes to the same file, and it therefore requires a lot of synchronisation. Using the co-design approach, we will carefully remove synchronisation on most operations while retaining a semantics reasonably similar to Posix. However, our preliminary CISE analysis shows that, in some corner cases, `rename` conflicts with concurrent `rename` operations. It follows that, either `rename` must be synchronised, or anomalies are unavoidable, such as loss or duplication of whole directories.

User-friendly, robust analysis tool. Our prototype CISE Tool is sufficiently advanced to prove the concept. It already checks the CISE Rules to prove the safety of the application. If not, it reports a counter-example that the developer can inspect to fix the problem [55, 56]. However, it is only a prototype and remains difficult to use. For instance, it requires writing a model of the application in the Z3 internal format. It is up to the developer to understand the counter-example, to deduce the issue, and to correct the application.

A short-term improvement will be to develop a translator from some high-level Domain-Specific Language (DSL) to Z3. Longer term, we would like the tool to leverage the application source code itself, either automatically or semi-automatically.

[1] For instance, replacing the atomic `rename` with a non-atomic copy to the new location, followed by deleting the source location.

Task 1		Application co-design				
Task Leader		Marc Shapiro, Inria Regal				
Deliverables	M12	User-friendly co-design tool				
	M24	Synchronisation Protocol Generator				
	M36	Co-design and proof of complete file system and other applications				
Participant		Inria Paris	Scality	CNRS-LIG	Télécom SP	**Total**
Person×months		45	8	10	6	69

Table 5: Summary of Task 1

An important objective is to automatically analyse the counter-examples, identifying conflicting pairs of operations.

Finally, we plan to work on extending the CISE logic. First, we wish to remove the assumption of causal consistency, and detect cases where causality is not required, in order to decrease overhead and increase parallelism. Second, we plan to extend CISE to relaxed forms of transactions such as Snapshot Isolation [14], PSI [73] or NMSI [62]. Finally, we wish to address testing. Tools such as symbolic execution [19] can systematically generate test cases from the source code itself, but suffer combinatorial explosion with parallel code. Our intuition is that an approach similar to CISE can make this quadratic, which would be a major breakthrough.

Automating protocol generation. The set of conflicting pairs from the previous step still needs to be translated into an efficient concrete protocol. An advanced tool would suggest efficient concurrency control protocols and/or weakenings of effect or invariant. The developer is still free to follow the suggestion, or to devise her own fix.

Operations that conflict must execute one after the other. For a given set of conflicting pairs, there are many possible implementations of this order. These implementations vary in efficiency and liveness. One extreme is to ensure simple mutual exclusion (e.g., ensuring each operation in a pair contains a read-modify-write instruction, or acquires and releases a lock). This is safe and live, but repeated synchronisation is unlikely to perform well. At the other extreme, we might allow only one process to execute conflicting operations (e.g., acquiring locks at the beginning of execution and never releasing them); this is safe, and much more efficient in terms of concurrency-control traffic; but it's unlikely to be live.

The best approach in terms of efficiency is often somewhere in the middle. The choice depends highly on the workload: for instance, if operations that conflict are always called by a single process, the "lock beforehand and never release" approach works well. A lazy approach, acquiring a lock on first use, and releasing it only when required by a different process, will probably give good results in many cases. Furthermore, there are more efficient alternatives to locking in some situations, e.g., leveraging causality or using escrow.

We envisage a tool based on heuristics to generate an efficient protocol, and to compare protocol behaviour under benchmark workloads or fault injection scenarios. The choice shall be guided by energy savings, latency, throughput, fault tolerance and other performance considerations, which depend on the workload and on the environment. The tool's heuristics should avoid deadlock, and the tool should verify liveness.

Detailed work programme

Subtask 1.1 *User-friendly, robust CISE Tool.* We plan to develop a front-end that parses the specification in a convenient Domain-Specific Language syntax (for instance, Z3lib or TLA+) before discharging the CISE Rules to the Z3 SMT solver. Later, we will consider methods for extracting the information directly from the source code of the application, either automatically or with assistance from the developer.

Subtask 1.2 *Generation of fixes for CISE violations.* This task will identify standard programming patterns that correct common subclasses of CISE violations. It will develop a heuristic tool that examines the counter-examples returned by the tool and will suggest fixes, either by weakening the application specification or by strengthening synchronisation.

Subtask 1.3 *Synchronisation protocol generation.* A number of standard concrete synchronisation protocols to avoid conflicts will be identified. We will compare their behaviours under benchmark workloads and fault scenarios. We will develop a tool to propose a suitable concrete protocol, according to runtime and environment

characteristics. Thanks to run-time monitoring and a feedback loop, the tool will be able to learn from experimental data and propose changes to the protocol when conditions change.

Success indicators Our current proof-of-concept CISE tool has been applied to few test cases. In RainbowFS, we aim to make it accessible to ordinary programmers, to leverage source code directly, to extend it to run-time testing and verification, and to apply it to the challenging file-system design described hereafter.

Therefore, we identify the following success indicators *(i)* The ability for ordinary programmers to efficiently use CISE tool thanks to a high-level DSL to co-design their applications and the suitable consistency guarentees. *(ii)* The characterization of most concrete synchronization protocols and their performance in application-specific workloads and faults scenarios. *(iii)* Identifying common patterns of CISE violations and guidelines on how to fix them.

Deliverables

M 12: User-friendly CISE co-design tool, translating a model written in a high-level language to the solver's format, discharging the proof obligations, and returning any counter-example in a high-level language.

M 24: Semi-automated synchronisation protocol generation. This tool will consider a limited set of synchronisation protocols and a number of well-specified workloads and scenarios.

M 36: Co-design and correctness proof of complete file-system design and an application using the file system.

Risks and risk mitigation The work proposed in Task 1 is ambitious but feasible. Indeed, the CISE logic and its proof of soundness have been published [31]; a prototype CISE analysis tool has been developed [55, 56] and demonstrated on some proof-of-concept examples, including a simplified model of the file system [54]. However, it is only a restricted research prototype, which is too complex and low-level for general use.

We see two main risks for this task. First, identifying standard programming patterns and heuristics to fix CISE violations might turn out to be too specific to applications. In this case, instead of the generation of fixes of CISE violation, we will provide a suite of use cases to help developers. Second, the performance of a concrete synchronisation protcol depends on workloads characteristics and fault scenarios. Both workloads and fault scenarios are dynamic. Hence, it might turn out to be difficult to devise benchmark workloads that are representative of real situations. In this case, we will at least provide guidelines for choosing synchronisation protocols, according to workload and runtime characteristics.

Task 2 : Modular geo-replication This task is summarised in Table 6.

Objectives of the task: *To provide the building blocks allowing developers to quickly implement an efficient and sound geo-replicated storage system/service with flexible (just-right) consistency semantics.*

The present task aims to help the developer transform an application specification, proved correct in the previous task, into a functional application skeleton. A skeleton comprises the core communication and data processing infrastructure. In particular, this includes the code that manages message transmission, reception and delivery, and enforces consistency invariants. This also includes the code that reacts to dynamic events such as faults or changing workloads; such events require to create or move replicas, in order to guarantee long-term availability and performance. Note that the consistency specification must define the behavior of the system in the presence of faults such as disconnection or crashes.

This task has two main challenges: *(i)* To provide lean and efficient primitive consistency components that can be combined to implement the application's communication and consistency requirements quickly. *(ii)* To provide a composition framework that ensures that the composition is as efficient as a monolithic implementation, in terms of message delays, number of messages, size of message, synchronisation steps, etc. To comply with the Just-Right Consistency approach, which is to "pay only for what you need," our modular approach should not impose any synchronisation or abstraction overhead not strictly required by the specification, and must use hardware resources efficiently.

In particular, the composed components should not impose any redundant steps, metadata, or spurious synchronisation. For instance, if two components both send a message with the same source and destination, the composition framework should consolidate them into a single message. In addition to protocol efficiency, the component model should make good use of the resources modern hardware, such as multicore CPUs and high-speed network interfaces.

Task 2		Modular geo-replication				
Task Leader		Vivien Quéma, CNRS LIG				
Deliverables	M12	Consistency components; deployment, profiling & monitoring tools				
	M24	Efficient composition; analysis & validation tools.				
	M36	Large-scale experimental validation of the component library & tools				
Participant		Inria Paris	Scality	CNRS-LIG	Télécom SP	**Total**
Person×months		27	8	40	18	93

Table 6: Summary of Task 2

Methods and technical decisions The key aspects of this tasks's approach is summarized as follows.

Modular decomposition of consistency and fault management. We propose to deconstruct consistency models along three (mostly-)orthogonal axes: *(i)* total order of operations relative to a single object; *(ii)* causal ordering of reads with respect to writes; and *(iii)* atomicity of transactions combining multiple operations. A consistency protocol can be strong along one axis and weak along another. Each axis corresponds a specific class of guaranteed invariants, respectively: constraints on the value of a single data item; partial order between events or data items; and equivalence between data items. Only total-order requires mutual synchronisation, and thus only the first axis is subject to the CAP trade-off. In contrast, atomicity and causality can be implemented with high availability.

Therefore, in order to improve the efficiency of application developers (time of development and safety of the resulting code), we will implement a library of consistency components. More precisely, there will be three classes of consistency components: one class for each of the three above-discussed axes, with several variants in each class (according to the chosen strength for the corresponding consistency dimension). To the best of our knowledge, the above-described "3-D" decomposition is new. We believe that it represents a good compromise between completeness and intelligibility.

This task will also study how modularity can be applied to fault-tolerance code. The idea is to design protocol abstractions enabling reuse, as much as possible, of the same fault-tolerance building blocks for different application-level consistency specifications. This is important because the fault-tolerance code is critical to both safety and performance, for instance, when operating in degraded mode or while rebuilding of a replica. Therefore, the design and implementation of correct and efficient fault-tolerant code is complex and time consuming and would strongly benefit from code reuse and simplified incremental development. To achieve this goal, we will build on our experience with the Abstract framework [8], for designing modular state machine replication protocols.

The design of the framework will be guided by *(i)* the opportunities of semi-automatic code/template generation from the toolchain of Task 1 and *(ii)* the requirements of the tools designed in this task. For these reasons, we plan to consider a clean-slate approach for the APIs of the framework, but we nonetheless intend to facilitate the development of many components by borrowing and adapting code from existing open-source projects. For example, we intend to borrow code from the G-DUR framework [6] (developed by two of the partners) and from CockroachDB [44] (an open-source clone of Google Spanner [21]).

Tools. This task will design and implement tools to assess the correctness and the efficiency of a skeleton built by composing library components. We envision two kinds of tools: *(i)* Checking correctness, i.e., that the application's safety and liveness; *(ii)* Check efficiency, i.e., that the implementation does not introduce unnecessary synchronisation or redundant steps. These tools will rely on a combination of static and dynamic approaches, leveraging the uniform design and API of the components.

We plan to leverage existing model-checking tools (e.g., CADP [39]) to assess the correctness of our implementations in fault-free cases. Similarly, we will extend existing stress-test tools, such as Jepsen [42], to assess our implementations when faults occur. Finally, we will develop simple tools to monitor the performance of the consistency protocols developed using our component library, collecting performance metrics such as message latency and throughput or space and time overheads.

Detailed work programme

Subtask 2.1 *Component library.* We will develop a modular library of consistency components and fault-tolerance components, with an efficient composition framework. The components offer fine grained modularity with respect to consistency and fault tolerance guarantees. Associated are tools to assess the soundness of the individual

components, the library glue code, and the resulting component assemblies.

Subtask 2.2 *Efficient composition framework.* This subtask will support the component library, both at the design level (i.e., the assembly of micro-protocols for consistency and fault-tolerance), and at the implementation level. For each component composition, our goal here it to reach the performance and resource footprint of a monolithic implementation. This will be supported by developing monitoring and profiling tools that spot performance bottlenecks.

Subtask 2.3 *Evaluation.* We will assess the benefits of Just-Right Consistency compared to existing monolithic approaches. This assessment feeds back iteratively into the design. Each iteration consists of an experimental campaign followed by the design and implementation of new optimizations. This feedback loop will make use of three kinds of workloads: *(i)* synthetic micro-benchmarks; *(ii)* existing applications (adapted from existing code); *(iii)* the file-system use case from Task 3.

Success indicators We see the following success criteria for this task: *(i)* The ability to reimplement existing geo-storage application designs (from the litterature) with low effort. *(ii)* For a given application design, the ability to achieve at least comparable performance with respect to a monolithic implementation. *(iii)* The ability to demonstrate that an implementation derived from the Just-Right-Consistency approach can yield tangible performance gains.

Deliverables

M 12: Library of consistency components; deployment, profiling and monitoring tools.

M 24: Efficient composition framework of consistency components; analysis and validation tools.

M 36: Large-scale experimental validation of the component library and tools.

Risks and risk mitigation The partners have preliminary experience towards this task. They have published preliminary deconstructions of consensus protocols [8] and of transactional protocols [6]. They have accumulated experience in tools for deployment and monitoring distributed systems [59, 60].

We see two main risks for this task. First, the degree of modularity that we envision for the components may turn out to introduce excessive complexity and/or a substantial performance overhead. In this case, we will restrict the modularity of the framework and consider only the consistency configurations that most useful for the file system use case studied in Task 3. Second, the monitoring and analysis that we envision may prove to be challenging to design and implement (e.g., due to heavy resource requirements). In this case, we might fall back on testing in a controlled environment.

Task 3 : Geo-replicated massive file system This task is summarised in Table 7.

Objectives of the task: *To design an entreprise-grade geo-replicated file system with consistency guarantees, and to demonstrate it at massive scale.*

The data storage market shows a growing demand for file and object based storage. This demand comes from the legacy use of file systems as one-size-fits-all storage solutions, together with a growing appetite for big data infrastructures (e.g., HDFS in Hadoop). Analyst firm IDC predicts the global file and object storage market will continue to gain momentum and reach $38 billion by 2017 [58]. Scality regularly gets requests for file systems up to hundred of petabytes.

Jointly to this demand for raw performance, compagnies are getting more and more de-centralised, operating at the geographical scale. However, as pointed out in II.4.5, existing file systems designs are not yet ready for Fog computing and geo-distribution. They either do not deliver convincing performance at that scale, or they still rely on a central metadata server. To bridge this technological gap and target a promising market, our flagship application in RainbowFS is a *geo-replicated massive file system*.[2] To build this file system in Task 3, we will address the following challenges:

- **Elastic Geo-distribution.** We provide a Fog-computing experience to the end-user. In particular, we target large-scale infrastructures that consist of dozens of large data centres (DCs) and possibly hundreds of smaller point-of-presence centres, scattered around the globe. This scale of infrastructure is required by today's digital businesses, which need distributed, always-on service; applications with extreme latency requirements will leverage replicas located very close to

[2] We also plan to apply it to several other real-world applications, for instance big-data analytics and indexing, outside of RainbowFS funding.

Task 3		Geo-replicated massive file system				
Task Leader		Vianney Rancurel, Scality				
Deliverables	M12	Single DC File System Prototype				
	M24	Geo-distributed Massive File System Prototype				
	M36	Large-scale Evaluation Report				
Participant		Inria Paris	Scality	CNRS-LIG	Télécom SP	**Total**
Person×months		36	26	10	30	102

Table 7: Summary of Task 3

their end users. The system shall support elastic scalability, dynamically and seamlessly adding and removing servers, or even whole DCs.

- **Massive Performance.** To achieve massive performance, we plan to leverage the fully-decentralized nature of the large-scale underlying infrastructure. Our file system will aggregate network bandwidth where possible, and will support massively parallel distributed operations. To ensure scalable geo-distribution, it will support partial replication.
- **Consistency, Security & Fault-tolerance.** Our file system must be dependable, ensuring consistency (as described under Task 1) and tolerating failures such as machine crashes, disconnection, or whole-DC failure. It shall be secure thanks to state-of-the-art cryptography techniques, and ensure integrity using erasure codes. An application should be able to have its own data placement policy, e.g., placing replicas in distinct locations to cope with the possibility of a DC being compromised, or ensuring that all its replicas remain within a given jurisdiction.

Methods and technical decisions Our technical decisions build upon our rich experience in designing and evaluating distributed data storage systems.

Synchronisation-free Operations Where Possible. We ensure our file system is both scalable and correct using the co-design approach developed in Task 1. The system maintains the invariant that both the directories and the internal (inode and block) data structures form a tree. We know from Task 1 that there is no need for strong synchronisation to implement the `add` and `remove` operations. However, the Posix standard contains additional operations that make it difficult to scale geographically and/or to high numbers of users and high update rates. We plan to make as many operations as possible synchronisation-free, without violating user-defined expectations, and to boost performance using techniques such as caching or close-to-open semantics [36]. For instance the referential-integrity invariant (every node is referenced from some directory) does not require synchronisation, only causal delivery of `create` and `delete` operations.

This methodology builds upon the CISE analysis and upon our previous geoFS design, a proof-of-concept geo-replicated file system [80]. In geoFS, all operations are asynchronous (concurrent `rename` suffers only a minor anomaly, i.e., duplication of a sub-tree, and concurrent updates to a file result in two renamed files).

Tailorable Data Consistency. Historically, file systems focused on the strong Posix model. More recent designs have shifted to ensuring availability under weaker consistency, because of the need for geo-distributed, planetary-scale and always-available services. However, existing file systems offer the same consistency level for *all* end-users and applications, and are often incorrect. RainbowFS departs from this monolithic vision of the file system, and explores fine-grained consistency, parameterised according to usage, while still ensuring correctness. Users may choose their semantics at the granularity of a sub-tree or even of a single file, thus supporting the *rainbow* of all reasonable consistency/available trade-offs. For instance, by user option, concurrent writes to a same file can either be disallowed as in NTFS, arbitrated by last-writer-wins as in NFS, or be merged similarly to SVN or git, if the file content is a CRDT, or cause the creation of two files side-by-side as in Microsoft OneDrive or geoFS. For `rename`, the user will have a choice of either occasional unavailability as in strongly-consistent systems, or occasional anomalies as in geoFS. Even if an application running above the tailorable file system chooses the more liberal options for availability and performance, it will remain safe, because it can itself be verified thanks to the tools from Task 1.

All the above options can be supported above a single, asynchronous codebase. To disallow a specific anomaly requires only some extra synchronisation, which can be added on top. In the limit, if all operations are made synchronous, the system exhibits the strong Posix semantics.

Object Store Everywhere. We build our file system atop an object store, currently being developed by Scality, code-named IrM hereafter. This object store is elastic, dependable and geo-distributed. It also supports data placement policies. RainbowFS implements all high-level file system calls as transactions of IrM-level operations. We cleanly separate data objects (file content) from metadata objects (inodes). The data store is optimised for write-once objects, and the metadata store for highly-mutable objects. Metadata forms a virtualisation layer above the object store to avoid update-in-place. This approach converges the file- and the object-based storage designs, which we believe is a worthwhile research topic on its own merit.

Fully-distributed Architecture. Current industrial-grade file systems [29, 70, 75] are built around a central metadata server. As the metadata server is a serialisation bottleneck, this design does not scale beyond the petabyte [71]. To support future sizes and geo-distributed Fog computing, our approach instead distributes metadata management across geo-distributed DCs. To support synchronisation when necessary inside the file system, the object store will support primitives such as read-modify-write (`rmw`) or compare-and-swap (`cas`). For instance, in the Posix semantics, creating a file is an atomic transaction that first creates an inode block, then adds the file to the parent directory with `rmw`. Only the latter operation needs strong consistency to ensures that two clients don't create the "same" file concurrently. Synchronisation and communication use the library of components developed in Task 2.

Experimental validation. As a reality check, we are planning a massive experimental deployment and benchmarking of the file system. It must be of sufficient scale to be credible and to be competitive with scientific publications in the best conferences, currently dominated by the American cloud giants such as Google and Amazon.

Our performance target aims for 10^5 servers over several DCs, supporting petabytes of data, and an update bandwidth exceeding 10 GB/s. However, a petabyte-scale experiment would be prohibitively expensive and would make sense only with real client data. To remain feasible within the budget of an ANR proposal, we aim for a terabyte-scale experiment, with 10^3 physical nodes and 10 servers per node, across five geographically-distributed DCs. The experiment plan is developed in more detail in Table 8 and Section III.3.

The experiment will initially use the French national Grid'5000 national experimentation facility [33], which can be accessed for free. Once the system has been thoroughly debugged on Grid'5000, and to expand beyond the Grid'5000 capacity, we will add resources rented from a commercial cloud provider. We expect to be able to execute essentially the same code on both platforms. Although only terabyte-scale, this setup should still enable us to draw meaningful conclusions on the way to the petabyte.

Detailed work programme

Subtask 3.1 *File System.* We design and implement a distributed file system with support for file replication, data redundancy and encryption, multi-scale fault-tolerance, and a rainbow of consistency levels. It is layered above the IrM object store. Intitially, it will support a single DC and a single level of consistency; later it will be extended to available geo-replication and tailorable consistency.

Subtask 3.2 *Evaluation.* This subtask is devoted to the extensive evaluation of our file system described above.

Success indicators We evaluate the success of Task 3 in terms of functionality and performance. This means that *(i)* We deliver a fully functional file system interface that can run industrial-grade benchmarks (e.g., IOzone, SPECsfs or FileBench [20, 52, 74]) as well as synthetic traces [86, 87]; *(ii)* the degree of consistency required by the user is satisfied at all time by the file system, from fully asynchronous as in geoFS, to fully-synchronous Posix semantics, while remaining correct; and *(iii)* the evaluation assesses that we reach the competitive performance in terms of throughput, latency, fault-tolerance, parallelism and geo-distribution.

Deliverables

M 12: Initial version of the prototype with base object and file operations at the level of a single DC. We use separate instances of the IrM object store to store data and metadata objects. The CISE tool verifies safety of the file system specification.

M 24: A more evolved prototype including the full support of file system operations in a single DC, and base operations across them. The underlying geo-distributed data block storage should support erasure coding, data locality as well as `put`/`get`/`rmw` operations.

M 36: Full geo-distributed evaluation of the RainbowFS prototype. This evaluation leverages resources taken from both Grid'5000 and commercial clouds.

Risks and risk mitigation While clustered file systems, such as NFS, Lustre or HDFS are widely deployed, peer-to-peer file systems [23, 53] never gained momentum. As a consequence, it is possible that in the rainbow of colors, very weakly consistent file systems seem risky to the end-user. To mitigate this problem, our insight is to avoid lost updates under weak consistency with the help of CRDT-like file operations. Another possible risk is related to the large-scale evaluation of our file system. Synthetic benchmarks should be as close as possible to realistic scenario. However, when a massive number of users concurrently use a file system, the number of reasonable scenarios to evaluate is too large. To avoid this problem, we plan to leverage public traces of personal cloud storages, e.g., Ubuntu One [32], as well as tools employed in the HPC community [66] to model I/O-intensive distributed applications.

III.2 Project organisation

The consortium and partners will be described in Section III.4.

The Project Leader is Marc Shapiro of Inria Regal. He will receive support for technical coordination from the post-doc, acting as coordinator.

The Management Board has the responsibility of general circulation of information, and technical and non-technical decisions that require coordination between the partners. This board includes the project leader, the coordinator, one representative per partner, and one representative per task. The same person can attend in multiple capacities. The board meets physically, by telephone or similar means, at least once a month.

Each partner names its representative in the Management Board at the start of the project. At the time of submitting the proposal, these are identified as Marc Shapiro of Inria, Vianney Rancurel of Scality, Vivien Quéma of CNRS-LIG, and Pierre Sutra of TSP.

The technical work described will be carried out by the different partners, organised in Tasks, under the leadership of a Task Leader, as described in Section III.1. A Task Leader may take the decisions required in the task. He shall inform in advance other tasks of any issues that impact them, and will take their input and requirements into account.

The project will use modern development tools, such as github, which support a distributed design and development team. Nonetheless, real communication requires physical meetings. There will be general meeting of all the project participants, at least once every six months. Smaller working groups will meet as required, either physically or electronically. Any of these meetings may invite outside expertise as required, as long as all the partners agree.

In case the Project Leader or one of the members of the Management Board does not wish to, or cannot fulfill his/her duties, the corresponding partner shall replace him or her, with the agreement of the remaining members of the Management Board. A Task Leader can be replaced, in similar circumstances, by a joint decision of the partners working on the task and of the Management Board.

In case of a conflict between partners, the Project Leader will endeavour to resolve the problem by consensus. For this task, he/she may consult outside experts. If consensus cannot be found, a special management board is called to vote on the issue; each partner institution has one vote; in addition the project leader has one vote and can resolve a tie.

III.3 Scientific explanation of funding request

The budget plan and funding request will be found in the administrative submission document. More detail follows.

We request funding for three PhD students, one post-doctoral researcher, and one junior engineer, all full time. The students are involved in all the tasks, but one is more specifically responsible of the co-design task, one of the modular consistency task, and one of the file system task. Although the post-doc is assigned to the Inria budget, he or she will be serving the whole project. He or she will focus on the modular consistency challenge, and will also coordinate the research across the different areas of the project. The engineer will work on connecting the file system with the underlying local storage and replication substrate, contribute to the massive file system experiment, and help integrate the project's results into Scality's products. The partners will also bring in other students and engineers, funded from other sources, who will help with the project.

Publication on experimental research in distributed systems is currently dominated by large American cloud operators. European academia generally does not have the means to compete. To regain the research lead, our workplan includes an ambitious massive-scale experiment. It will leverage both Grid'5000 and commercial cloud resources, which would be impossible without the requested ANR funding.

Its target is to experiment with 1 100 nodes, 10 servers per node, across five geographically-distributed data centres (DCs). Preparation and ramp-up will be done

Total hours (Total period):	720						
For setup:	72						
For experiments:	648						
Replication Factor in AW	2	(Each data item will be replicated in 3 DCs: 2 AWS + 1 Grid5000)					
Total DCs in AWS:	3	(Plus two DCs in Grid5000)					

Node Costs

Node Type	Amount/DC	Amount total	Type	Avg cost/h/node	Avg cost/period/node	Total cost/period	
File and object shard server	270	810	c3.large	0.105	75.6	61236	
Monitoring	3	9	c3.large	0.105	75.6	680.4	
Analysis	1	3	c3.xlarge	0.21	151.2	453.6	
Load generators	26	78	c3.xlarge	0.21	151.2	11793.6	
Sum Nodes	300	900		0.63	453.6	74163.6	

Network Costs

Traffic Type	GB/h/node/replica	GB/h/node	GB/h per DC	GB/h total	Cost/GB	Cost/period/DC	Total cost/period
Inter DC Traffic Outbound	5	5	1350	4050	0.02	17496	52488
Inter DC Traffic Inbound	5	5	1350	4050	0	0	0
Sum Traffic	10	10	2700	8100		17496	52488

Storage Costs

Storage Type		GB per DC	GB total	Cost/month/GB	Cost/Period/DC	Total cost/period
Image/data storage		2000	6000	0.05	100	300

Total Sum (USD)					179439.6
Total Sum (€) - Approximation	exchange rate:	0.887592			159269.2

Table 8: Budget of large-scale experiment

on the Grid'5000 national facility [33]. Optimistically we hope to be able to run two DCs on Grid'5000; this is already stretching the upper limit of what Grid'5000 can provide. To scale beyond, we must rent resources from a commercial cloud provider.

Our experiment plan and budget is detailed in Table 8. For cost and feasibility reasons we aim for a terabyte-scale experiment; a petabyte-scale experiment would be prohibitively expensive and would make sense only with real client data. We expect an experiment duration of 720 hours, spread over two or three months. We save on setup time by previously debugging the deployment in Grid'5000. We will provision two DCs in Grid'5000, and rent virtual machines in three DCs of a commercial cloud provider. The replication factor is three, i.e., on average there will be one replica of every data item in Grid'5000 and two in the commercial cloud.

Each Grid'5000 DC will consist of 100 file server nodes, and 10 nodes for load generation and monitoring; this is an optimistic estimate, on the hope that Grid'5000 will grow, because today it is extremely difficult to reserve even 100 nodes at a time in Grid'5000. Each DC in the commercial cloud will consist of 300 medium-size nodes, of which 270 for file system servers, and 30 for load generation and monitoring. Furthermore, we estimate 5 GB/hour/node/replica of inter-DC traffic in each direction, and 2 TB of image and data storage per DC. To estimate the costs, Table 8 uses the current AWS list prices as our data point; the total comes to 160 k€. The commercial cloud provider will be selected on price and functionality; all other things being equal, a European provider such as OVH will be preferred. This amount is budgeted to CNRS LIG, who will be co-ordinating the experiment.

Inria Paris (Regal) requests funding for a postdoc and a PhD student. The postdoc will focus his or her research on Tasks 1 and 2, and will help with the co-design of the file system application. He or she will also assist the Project Leader in the coordination of technical and administrative tasks. The research of the PhD student will focus on modular consistency and application co-design. The senior researcher's research workplan is as follows. Dr. Shapiro leads the project and focuses on Task 1. Prof. Monnet focuses on Task 2. Dr. Makpangou focuses on Task 3.

As project coordinator, with a high manpower investment, Regal requests 45 k€ to attend project meetings and scientific conferences, and 10 k€ for computing equipment.

CNRS LIG The cost of the large-scale Cloud-based experiments, detailed above, is budgeted to CNRS LIG, who co-ordinates the experiments. CNRS LIG also requests funding for a PhD student who will mostly work on Task 2. The student will be in charge of designing

the component library and its associated monitoring tools. He will also contribute to Task 3.

In addition, CNRS LIG requests 20 k€ for travel (to attend project meetings and scientific conferences) and 5 k€ to acquire laptops for the design and development work. The slightly higher travel budget (compared to other partners with similar manpower) is explained by the fact that CNRS LIG is the only partner located outside the Paris area (where most project meetings will be held).

Télécom SudParis (TSP) will work mostly on the development and evaluation of the geo-distributed massive file system. To help in the coding of the file system and the large-scale evaluation, TSP requests the funding of a PhD student. Besides that, the funding demand of TSP also includes 15 k€ for travel to project meetings and scientific conferences, as well as 5 k€ to be spent in small equipment.

Scality In order to participate to the scientific research and to prototype the scientific results of RainbowFS into their product line, Scality will take the opportunity of this project to hire a permanent-staff (CDI) research engineer. The engineer will participate in the design and development of the geo-distributed file system (Task 3) above Scality's object storage system. He or she will also help to develop and put to use the modular consistency, deployment, monitoring and analysis components studied in Task 2. (Regarding the co-design tools studied in Task 1, Scality acts mostly as users.) This is important, because practical implementation research serves to validate the scientific model, and can provide unanticipated insights. This is also beneficial for the usefulness of the project since it will be integrated into Scality's product.

III.4 Project partners

III.4.1 Consortium as a whole

The partners together have the expertise and the critical mass necessary for this project. They already have a history of collaboration, including design and development of a highly-available file system [65, 80] and of advanced consistency protocols [6, 62, 76–78].

Scality is world leader in massive distributed storage on commodity system architectures. Inria Regal brings expertise in large-scale replication, different consistency models, and CRDTs. CNRS LIG are recognised experts in fault-tolerance in distributed systems using replication protocols (both within a datacenter and across datacenters). Télécom SudParis are experts in consistency protocols and distributed storage systems; they contributed to the design and implementation of Infinispan [51], the flagship distributed key-value store developped by RedHat.

III.4.2 Principal Investigator

The Principal Investigator is Marc Shapiro, a Senior Researcher *(Directeur de recherche)* at Inria and LIP6. He is a leading European expert in distributed computing. He led the SOS group (*Systèmes d'objets répartis*) at Inria for 13 years, then the Cambridge Distributed Systems Group at Microsoft Research Cambridge for 6.5 years.

He has participated in numerous cooperative projects, with both academic and industrial partners. He was the leader of the recent ANR project ConcoRDanT. He is currently (until Sept. 2016) the Principal Investigator for the SyncFree FP7 project on ultra-scalable techniques for sharing mutable data in wide-area end-user application scenarios. The outcome of both projects will be used in RainbowFS.

He authored 89 international publications, 35 invited talks, 18 recognised software systems, and five patents. Recent relevant publications include POPL 2016 [31], EuroSys 2015 [10], PaPoC 2015 [22, 85], Systor 2015 [80], SRDS 2015 [11], Middleware 2014 and 2015 [6, 93], SIGACT News 2014 [40], OPODIS 2014 [3], SRDS 2013 [62], and SSS 2011 [68], all available online.

III.4.3 Inria Paris (Projet Regal)

Dr. Makpangou, Prof. Monnet and Dr. Shapiro are members of Regal, a joint research group of Inria Paris with CNRS and Université Pierre et Marie Curie (Paris 6) through the Laboratoire d'Informatique de Paris 6, LIP6 (UMR 7606).

The research of the Regal team addresses the theory and practice of Computer Systems, including multicore computers, clusters, networks, peer-to-peer systems, cloud computing systems, and other communicating entities such as swarms of robots. It addresses the challenges of communicating, sharing information, and computing correctly in such large-scale, highly dynamic computer systems. This includes addressing the core problems of communication, consensus and fault detection, scalability, replication and consistency of shared data, information sharing in collaborative groups, dynamic content distribution, and multi- and many-core concurrent algorithms.

Regal has developed a number of systems, such as SwiftCloud [93], G-DUR [6], NMSI [62], CRDTs [68, 69], and the CISE Tool [31, 56] that are relevant to this proposal.

III.4.4 CNRS Laboratoire d'Informatique de Grenoble (LIG) The Erods team at LIG is working in the domains of operating systems and distributed systems from both a practical and theoretical perspective. Recent works of the team include efficient protocols for the design of fault-tolerant distributed systems (e.g. state-machine replication protocols), optimization techniques for systems executed on NUMA multicore machines (e.g. thread and memory placement algorithms), as well as profiling and monitoring tools for both centralized and distributed systems (e.g. the MemProf memory profiler).

Vivien Quéma will be the technical leader for the CNRS-LIG partner. He is Professor of Computer Science at Grenoble INP. He has been a visiting researcher at the University of Rome 1 "La Sapienza," at EPFL, at UT Austin, and at LIP6 (Paris 6). He has been working for ten years on the design and optimization of complex systems of various scales, including multicore operating systems, replicated servers and large-scale peer-to-peer systems. He has co-authored research papers in venues such as ACM TOCS, OSDI, ASPLOS, EuroSys, USENIX ATC, DSN, Middleware, ICDCS and SRDS. He received a best paper award at EuroSys 2010 for his work on modular fault-tolerant replication protocols and a best paper award at USENIX 2015 for his work on performance optimization of NUMA multicore systems.

III.4.5 Télécom SudParis The ACMES team of Télécom SudParis focuses on the design of algorithms, components and services for distributed systems, from small to large scale. The team is part of the SAMOVAR Laboratory (UMR 5157). Recent advances in the team include a multiscale framework for the IoT, concurrency control algorithms for multicore architectures, as well as a middleware of distributed objects.

Pierre Sutra (assistant professor) will be in charge of the Télécom SudParis partner. Recently, Pierre was a junior researcher at the University of Neuchâtel, and a post-doc at INRIA. Pierre received his PhD from UPMC in 2010. He is interested in both the theory and practice of distributed systems, with an emphasis on data consistency and concurrency. His recent publications on the topic of data storage systems and concurrency control include DISC 2015 [18], CLOUD 2015 [59], Middleware 2014 [93], SRDS 2014 [28, 35] and OPODIS 2014 [79].

III.4.6 Scality S.A. Founded in 2009, Scality is a French company with headquarters in San Francisco and subsidiaries in Japan and France. Most of the R&D and engineering is done in France, where Scality employs 102 people. Scality is an industry leader in software-defined storage at petabyte scale. Scality's customers include four of the top ten cable operators in the US, the second largest telco in France, leading operators in Japan, leading television network in Germany, and the second largest online video site in the world.

Scality deploys software-based storage solutions that deliver billions of files to more than two hundred million users daily, with 100% availability. At petabyte scale, hardware-based scale-up approaches are very expensive and also very hard to scale and operate. Scality's answer is a 100% software-based, scale-out approach, using a cluster of commodity servers abstracted by a logical layer that handles data access, data durability, data availability and data movement. It supports a Posix-compliant distributed filesystem with high-level APIs (FUSE, NFS, CIFS and CDMI), on top of a Distributed Hash Table (DHT).

IV Impact of project, strategy for take-up, protection and exploitation of results / Impact du projet, stratégie de valorisation, de protection et d'exploitation des résultats

IV.1 Impact

The project will have scientific, practical and economic impact. Let's consider them in reverse order.

IV.1.1 Economic Impact In a cloud-scale system, the choice of the consistency model is a difficult and vexing choice. Strongly-consistent storage shields the developer from the complexities of parallelism and distribution, but comes at a stiff price. A highly available system can be faster and cheaper, but early adopters learned the hard way that the anomalies of weak consistency are error-prone. Pragmatically, in real-world distributed systems, weak and strong consistency coexist, but this combination is hard to understand.

Our approach enables the developer to identify the weakest, most available and most efficient synchronisation pattern that supports her application. This gives an economic advantage, because weak consistency consumes considerably less resources and responds more

quickly than strong consistency, and because the developer will spend less time debugging than with a manual approach. This opens the market of cloud-scale applications to newcomers, enabling European SMEs to compete with the American giants.

Scality SA will be the first to benefit. They expect that the file system designed in this project to bring in sizeable revenue, thanks to its performance, scalability and novel features. It will also help Scality's customers to transition smoothly from files to object storage.

IV.1.2 Practical Impact Our approach builds upon a recent breakthrough, the CISE logic, which makes it possible to prove, with only polynomial complexity, that a given application specification satisfies given integrity invariants above a given consistency model. Furthermore, if the analysis shows the application to be incorrect, it returns a counter-example, which guides the developer towards a fix. She can either weaken the application specification or strengthen the consistency.

This is a complete change from current practices to distributed software development. Classical approaches proceed from a predefined consistency model, and leave it to the developer to reason about application correctness. In contrast, our approach ensures that a specific application, running on a specific consistency semantics, is safe with respect to specific properties relevant to the application. This lets the developer focus on her problem, and gives greater confidence that the application is correct.

Debugging a distributed application is extremely complex, because of the combinatorics of parallel execution, non-deterministic scheduling and messaging, and failures. The current practice is mostly based on trial and error. Using the CISE logic, we are hopeful to be able to generate systematic test cases for distributed systems with only quadratic complexity.

IV.1.3 Scientific Impact A successful RainbowFS project will help disseminate knowledge about the CISE logic, a very recent breakthrough. Furthermore, our research aims to improve the existing tool and logic in several directions, making them more widely applicable. These directions include adding a causality analysis, supporting common snapshot-based forms of transactions, and applying the logic, not only to static analysis, but also to testing and debugging.

Currently, experimental research in distributed systems is dominated by large American cloud operators. European academia generally does not have the means to compete. To regain the research lead, our workplan includes an ambitious massive experiment, leveraging Grid'5000 and massive commercial cloud resources, which would be impossible without the requested ANR funding.

IV.2 How the project addresses the challenges

This section refers back to the challenges listed in Section II.5. Our agenda addresses scientific and technical issues related to some of the hurdles identified for Axis 7, namely: *(i)* issues regarding scaling up in various dimensions, *(ii)* a far-reaching change to the architecture and operation of infrastructures, *(iii)* reliability and fault resilience. RainbowFS is completely in line with the challenges put forth by ANR in Research Axis 7: *"aim to avoid isolated infrastructures geared to just one type of application", "using an agile approach to comply with future, often unanticipated, developments", "infrastructures must be capable of achieving high levels of performance and efficiency, while being open and agile so that they can be adjusted to meet the diverse, dynamic requirements of the various application categories."*

By adapting consistency protocols to application needs, we provide the ability to offer efficient data access to a wide variety of applications, on a large scale. Enabling safe access to weak consistency has the potential to reduce substantially both the monetary and the energy cost for large-scale distributed applications.

Beyond the concerns of the ANR Strategic Programme discussed in Section II.5, the RainbowFS proposals addresses major concerns of the scientific community and industry.

IV.3 Scientific communication and uptake

The members of the project plan to use the tools and the library components (developed in Tasks 1 and 2 of the project) as a teaching vehicle for lectures and hands-on sessions in distributed systems and distributed algorithms (all the academic participants in the project are already strongly involved in the teaching staff of graduate-level classes related to distributed systems and algorithms in several French universities). The teaching material (source code, companion documents, problem sets and answers) will be made freely available to the teaching community at large via a dedicated website.

The project leader, Marc Shapiro, is a member of ACM's Distinguished Speaker Program and will leverage this opportunity to present the results of the project to various (academic and industrial) audiences worldwide.

IV.4 Intellectual property and exploitation of results

All the partners, both academic and industrial, plan to make the software and other artefacts developed in this project available as open source. We plan to publish scientific results in major conferences and journals related to operating systems, distributed and parallel systems, storage systems, and related areas.

In addition to the usual scientific papers, we also plan to promote the results and the software produced by the projects through additional channels: *(i)* We will propose a tutorial session to a major international conference and/or a thematic school; *(ii)* We will post tutorial videos on YouTube or similar websites, similarly to the CISE demo already on YouTube [55].

IV.5 Industrial exploitation

The project follows on an existing scientific collaboration around geoFS between Regal and Scality [80], and will help transfer the CISE tools developed by Inria Regal [12, 54–56] to Scality.

By ensuring guarantees, the RainbowFS approach helps to develop novel applications easily without specialised distributed systems expertise. This lowers the cost of entry and creates opportunities for the startup ecosystem. By minimising synchronisation, this also enables applications to run efficiently in commodity clouds or clusters and achieve internet-size scalability and fast growth. With previous approaches, these two goals conflict.

Our approach will be applied to the development of a file system, which has high economic value on its own merit. Scality, a French SME whose development and R&D teams are in France, is a world leader in massive storage on commodity system architectures, an area of great economic importance. The RainbowFS project aims both to advance the scientific state of the art in distributed systems and to enlarge Scality's offerings. Scality plans to include the file system in its offerings within the timeframe of the project.

The RainbowFS technology will allow Scality to build a file system on top of their object storage system. It will support legacy applications, because the specific consistency requirements needed by the application will be proven. In addition, its versatile consistency semantics, tunable to application requirements, will appeal to a wider range of customers and market segments. Furthermore, our verification tools will enable customers to check correctness. These features will provide Scality with a significant competitive advantage.

Scality believes that the file system will provide a very interesting offer, in line with current market demand for storage technologies allowing a smooth transition from a file-based storage model to an object-based storage model. Scality currently estimates that this transition will last around ten years, and corresponding market to be approximately USD 100 billion. Thanks to the RainbowFS technology, Scality hopes to "unlock" and lead this market, and strengthen its product offer on top of the Scality object store, while comforting its leader role in the object storage world.

References / Références bibliographiques

[1] D. J. Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 45(2):37–42, Feb. 2012.

[2] A. Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. PhD thesis, Mass. Institute of Technology, Cambridge, MA, USA, Mar. 1999.

[3] M. K. Aguilera, L. Querzoni, and M. Shapiro, editors. *Principles of Distributed Systems*, volume 8878 of *Lecture Notes in Comp. Sc.*, Cortina d'Ampezzo, Italy, Dec. 2014.

[4] D. D. Akkoorath, A. Tomsic, M. Bravo, et al. Cure: Strong semantics meets high availability and low latency. In *Int. Conf. on Distributed Comp. Sys. (ICDCS)*, Nara, Japan, 2016.

[5] P. Alvaro, N. Conway, J. Hellerstein, et al. Consistency analysis in Bloom: a CALM and collected approach. In *Biennial Conf. on Innovative DataSystems Research (CIDR)*, Asilomar, CA, USA, Jan. 2011.

[6] M. S. Ardekani, P. Sutra, and M. Shapiro. G-DUR: A middleware for assembling, analyzing, and improving transactional protocols. In *Int. Conf. on Middleware (MIDDLEWARE)*, pp. 13–24, Bordeaux, France, Dec. 2014.

[7] H. Attiya, F. Ellen, and A. Morrison. Limitations of highly-available eventually-consistent data stores. In *Symp. on Principles of Dist. Comp. (PODC)*, pp. 385–394, Donostia-San Sebastián, Spain, July 2015.

[8] P.-L. Aublin, R. Guerraoui, N. Knežević, et al. The next 700 BFT protocols. *Trans. on Computer Systems*, 32(4): 12:1–12:45, Jan. 2015. ISSN 0734-2071.

[9] P. Bailis, A. Davidson, A. Fekete, et al. Highly available transactions: Virtues and limitations. *Proc. VLDB Endow.*, 7 (3):181–192, Nov. 2013.

[10] V. Balegas, N. Preguiça, R. Rodrigues, et al. Putting consistency back into eventual consistency. In *Euro. Conf. on Comp. Sys. (EuroSys)*, pp. 6:1–6:16, Bordeaux, France, Apr. 2015.

[11] V. Balegas, D. Serra, S. Duarte, et al. Extending eventually consistent cloud databases for enforcing numeric invariants. In *Symp. on Reliable Dist. Sys. (SRDS)*, pp. 31–36, Montréal, Canada, Sept. 2015.

[12] V. Balegas, C. Li, M. Najafzadeh, et al. Geo-replication: Fast if possible, consistent if necessary. *IEEE Data Engineering Bulletin*, 2016.

[13] Basho Inc. Riak distributed database. http://basho.com/riak/, 2016.

[14] H. Berenson, P. Bernstein, J. Gray, et al. A critique of ANSI SQL isolation levels. *SIGMOD Rec.*, 24(2):1–10, May 1995. ISSN 0163-5808.

[15] D. Bermbach and S. Tai. Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior. In *W. on Middleware for Service Oriented Computing*, p. 1, Lisbon, Portugal, Dec. 2011.

[16] K. Birman, G. Chockler, and R. van Renesse. Toward a Cloud Computing research agenda. *ACM SIGACT News*, 40 (2):68–80, June 2009.

[17] F. Bonomi, R. Milito, J. Zhu, et al. Fog computing and its role in the internet of things. In *W. on Mobile Cloud Computing*, pp. 13–16, Helsinki, Finland, 2012.

[18] Z. Bouzid, M. Raynal, and P. Sutra. Brief Announcement: Anonymous Obstruction-free (n, k)-Set Agreement with $n - k + 1$ Atomic Read/Write Registers. In *DISC 2015*, volume LNCS 9363 of *29th International Symposium on Distributed Computing*, Tokyo, Japan, Oct. 2015.

[19] C. Cadar, D. Dunbar, and D. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Symp. on Op. Sys. Design and Implementation (OSDI)*, pp. 209–224, San Diego, CA, USA, 2008.

[20] D. Capps. IOzone filesystem benchmark. www.iozone.org, July Retrieved March 2016.

[21] J. C. Corbett, J. Dean, M. Epstein, et al. Spanner: Google's globally-distributed database. In *Symp. on Op. Sys. Design and Implementation (OSDI)*, pp. 251–264, Hollywood, CA, USA, Oct. 2012.

[22] T. Crain and M. Shapiro. Designing a causally consistent protocol for geo-distributed partial replication. In *W. on Principles and Practice of Consistency for Distr. Data (PaPoC)*, co-located with EuroSys 2015, Bordeaux, France, Apr. 2015.

[23] F. Dabek, M. F. Kaashoek, D. Karger, et al. Wide-area cooperative storage with CFS. In *Symp. on Op. Sys. Principles (SOSP)*, pp. 202–215, Banff, Alberta, Canada, 2001.

[24] A. Davies and A. Orsaria. Scale out with GlusterFS. *Linux J.*, 2013(235), Nov. 2013. ISSN 1075-3583.

[25] G. DeCandia, D. Hastorun, M. Jampani, et al. Dynamo: Amazon's highly available key-value store. In *Symp. on Op. Sys. Principles (SOSP)*, volume 41 of *Operating Systems Review*, pp. 205–220, Stevenson, Washington, USA, Oct. 2007.

[26] J. Du, S. Elnikety, and W. Zwaenepoel. Clock-SI: Snapshot isolation for partitioned data stores using loosely synchronized clocks. In *Symp. on Reliable Dist. Sys. (SRDS)*, pp. 173–184, Braga, Portugal, Oct. 2013.

[27] R. Escriva, B. Wong, and E. G. Sirer. HyperDex: A distributed, searchable key-value store. In *SIGCOMM*, pp. 25–36, Helsinki, Finland, 2012.

[28] P. Felber, M. Pasin, E. Rivière, et al. On the support of versioning in distributed key-value stores. In *33rd IEEE International Symposium on Reliable Distributed Systems, SRDS 2014, Nara, Japan, October 6-9, 2014*, pp. 95–104, 2014.

[29] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Symp. on Op. Sys. Principles (SOSP)*, pp. 29–43, Bolton Landing, NY, USA, Oct. 2003.

[30] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002. ISSN 0163-5700.

[31] A. Gotsman, H. Yang, C. Ferreira, et al. 'Cause I'm strong enough: Reasoning about consistency choices in distributed systems. In *Symp. on Principles of Prog. Lang. (POPL)*, pp. 371–384, St. Petersburg, FL, USA, 2016.

[32] R. Gracia Tinedo, T. Yongchao, J. Sampe, et al. Dissecting ubuntuOne: Autopsy of a global-scale personal Cloud back-end. In *IMC 2015, ACM Internet Measurement Conference, October 28-30, 2015, Tokyo, Japan*, Tokyo, JAPON, 10 2015.

[33] Grid'5000. Grid'5000 Home. https://www.grid5000.fr/, retrieved March 2016.

[34] R. Guerraoui, R. R. Levy, B. Pochon, et al. Throughput optimal total order broadcast for cluster environments. *ACM Trans. Comput. Syst.*, 28(2):5:1–5:32, July 2010. ISSN 0734-2071.

[35] R. Halalai, P. Sutra, E. Rivière, et al. ZooFence: Principled service partitioning and application to the ZooKeeper coordination service. In *Symp. on Reliable Dist. Sys. (SRDS)*, pp. 67–78, Nara, Japan, Oct. 2014.

[36] J. H. Howard, M. L. Kazar, S. G. Menees, et al. Scale and performance in a distributed file system. *Trans. on Computer Systems*, 6(1):51–81, Feb. 1988.

[37] F. Hupfeld, T. Cortes, B. Kolbeck, et al. The XtreemFS architecture — a case for object-based file systems in grids. *Concurr. Comput.: Pract. Exper.*, 20(17):2049–2060, Dec. 2008. ISSN 1532-0626.

[38] IEEE and The Open Group. Standard for information technology — Portable Operating System Interface — system interfaces, 2013.

[39] INRIA CONVECS team. CADP: Construction and analysis of distributed processes. http://cadp.inria.fr, 2016.

[40] B. Kemme, A. Schiper, G. Ramalingam, et al. Dagstuhl seminar review: Consistency in distributed systems. *SIGACT News*, 45(1):67–89, Mar. 2014. ISSN 0163-5700.

[41] K. Kingsbury. Jepsen blog. https://aphyr.com/tags/Jepsen, 2016.

[42] K. Kingsbury. Jepsen: Breaking distributed systems so you don't have to. https://github.com/aphyr/jepsen, 2016.

[43] R. Klophaus. Riak Core: building distributed applications without shared state. In *Commercial Users of Functional Programming (CUFP)*, pp. 14:1–14:1, Baltimore, Maryland, 2010.

[44] C. Labs. CockroachDB: A scalable, survivable, strongly-consistent SQL database.

[45] R. Ladin, B. Liskov, L. Shrira, et al. Providing high availability using lazy replication. *Trans. on Computer Systems*, 10 (4):360–391, 1992.

[46] L. Lamport. The part-time parliament. *Trans. on Computer Systems*, 16(2):133–169, May 1998.

[47] L. Lamport. Lower bounds for asynchronous consensus. In *Future Directions in Distributed Computing*, pp. 22–23. Springer-Verlag, 2003.

[48] C. Li, D. Porto, A. Clement, et al. Making geo-replicated systems fast as possible, consistent when necessary. In *Symp. on Op. Sys. Design and Implementation (OSDI)*, pp. 265–278, Hollywood, CA, USA, Oct. 2012.

[49] C. Li, J. Leitão, A. Clement, et al. Automating the choice of consistency levels in replicated systems. In *Usenix Annual Tech. Conf.*, pp. 281–292, Philadelphia, PA, USA, June 2014.

[50] P. Mahajan, L. Alvisi, and M. Dahlin. Consistency, availability, and convergence. Technical Report UTCS TR-11-22, Dept. of Comp. Sc., The U. of Texas at Austin, Austin, TX, USA, 2011.

[51] F. Marchioni and M. Surtani. *Infinispan Data Grid Platform*. Packt Publishing Ltd, 2012.

[52] J. Mauro, S. Shepler, and V. Tarasov. FileBench. sourceforge.net/projects/filebench, Retrieved March 2016.

[53] A. Muthitacharoen, R. Morris, T. M. Gil, et al. Ivy: a read/write peer-to-peer file system. In *Symp. on Op. Sys. Design and Implementation (OSDI)*, volume 36 of *Operating Systems Review*, pp. 31–44, Boston, MA, USA, Dec. 2002.

[54] M. Najafzadeh. *The Analysis and Co-design of Weakly-Consistent Applications*. PhD thesis, Université Pierre et Marie Curie (UPMC), Paris, France, Apr. 2016.

[55] M. Najafzadeh and M. Shapiro. Demo of the CISE tool, Nov. 2015. https://youtu.be/HJjWqNDh-GA.

[56] M. Najafzadeh, A. Gotsman, H. Yang, et al. The CISE tool: Proving weakly-consistent applications correct. Rapp. de Recherche RR-8870, Institut National de la Recherche en Informatique et Automatique (Inria), Rocquencourt, France, Feb. 2016.

[57] D. Perkins, N. Agrawal, A. Aranya, et al. Simba: Tunable end-to-end data consistency for mobile apps. In *Euro. Conf. on Comp. Sys. (EuroSys)*, p. ???, Bordeaux, France, Apr. 2015.

[58] A. Potnis, N. Yezhkova, and A. Nadkarni. Worldwide file- and object-based storage 2014–2018 forecast. Market Analysis 251626, IDC, Framingham, MA, USA, oct 2014.

[59] D. L. Quoc, C. Fetzer, P. Felber, et al. Unicrawl: A practical geographically distributed web crawler. In *8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27 - July 2, 2015*, pp. 389–396, 2015.

[60] D. Reddy Malikireddy, M. Saeida Ardekani, and M. Shapiro. Emulating geo-replication on Grid'5000. Technical Report RT-455, Institut National de la Recherche en Informatique et Automatique (Inria), Paris and Rocquencourt, France, Aug. 2014.

[61] S. Roy, L. Kot, G. Bender, et al. The Homeostasis protocol: Avoiding transaction coordination through program analysis. In *Int. Conf. on the Mgt. of Data (SIGMOD)*, pp. 1311–1326, Melbourne, Victoria, Australia, 2015.

[62] M. Saeida Ardekani, P. Sutra, and M. Shapiro. Non-Monotonic Snapshot Isolation: scalable and strong consistency for geo-replicated transactional systems. In *Symp. on Reliable Dist. Sys. (SRDS)*, pp. 163–172, Braga, Portugal, Oct. 2013.

[63] N. Schiper, P. Sutra, and F. Pedone. P-Store: Genuine partial replication in wide area networks. In *Symp. on Reliable Dist. Sys. (SRDS)*, pp. 214–224, New Dehli, India, Oct. 2010.

[64] P. Schwan. Lustre: Building a File System for 1,000-node Clusters. In *Proceedings of the 2003 Linux Symposium*, July 2003.

[65] M. Segura, V. Rancurel, V. Tao, et al. Scality's experience with a geo-distributed file system. In *Posters & Demos Session, Int. Conf. on Middleware (MIDDLEWARE)*, Middleware Posters and Demos '14, pp. 31–32, Bordeaux, France, Dec. 2014.

[66] H. Shan, K. Antypas, and J. Shalf. Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pp. 42:1–42:12, Piscataway, NJ, USA, 2008.

[67] M. Shapiro, N. Preguiça, C. Baquero, et al. A comprehensive study of Convergent and Commutative Replicated Data Types. Rapp. de Recherche 7506, Institut National de la Recherche en Informatique et Automatique (Inria), Rocquencourt, France, Jan. 2011.

[68] M. Shapiro, N. Preguiça, C. Baquero, et al. Conflict-free replicated data types. In *Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 6976 of *Lecture Notes in Comp. Sc.*, pp. 386–400, Grenoble, France, Oct. 2011.

[69] M. Shapiro, N. Preguiça, C. Baquero, et al. Convergent and commutative replicated data types. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, (104):67–88, June 2011.

[70] K. Shvachko, H. Kuang, S. Radia, et al. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pp. 1–10, May 2010.

[71] K. V. Shvachko. HDFS Scalability: The Limits to Growth. *USENIX login*, 35(2), Apr. 2010.

[72] A. Sousa, R. Oliveira, F. Moura, et al. Partial replication in the database state machine. In *Int. Symp. on Network Comp. and App. (NCA'01)*, pp. 298–309, Cambridge MA, USA, Oct. 2001.

[73] Y. Sovran, R. Power, M. K. Aguilera, et al. Transactional storage for geo-replicated systems. In *Symp. on Op. Sys. Principles (SOSP)*, pp. 385–400, Cascais, Portugal, Oct. 2011.

[74] SPEC. SPECsfs2014. www.spec.org/sfs2014, Retrieved March 2016.

[75] Sun Microsystems, Inc. NFS: Network File System protocol specification. RFC 1094, Network Information Center, SRI International, Mar. 1989.

[76] P. Sutra and M. Shapiro. Fault-tolerant partial replication in large-scale database systems. In *Euro. Conf. on Parallel and Dist. Comp. (Euro-Par)*, pp. 404–413, Las Palmas de Gran Canaria, Spain, Aug. 2008.

[77] P. Sutra and M. Shapiro. Fast Genuine Generalized Consensus. In *Symp. on Reliable Dist. Sys. (SRDS)*, pp. 255–264, Madrid, Spain, Oct. 2011.

[78] P. Sutra, J. Barreto, and M. Shapiro. Decentralised commitment for optimistic semantic replication. In *Int. Conf. on Coop. Info. Sys. (CoopIS)*, Vilamoura, Algarve, Portugal, Nov. 2007.

[79] P. Sutra, E. Rivière, and P. Felber. A practical distributed universal construction with unknown participants. In *Int. Conf. on Principles of Dist. Sys. (OPODIS)*, volume 8878 of *Lecture Notes in Comp. Sc.*, pp. 485–500, Cortina d'Ampezzo, Italy, Dec. 2014.

[80] V. Tao, M. Shapiro, and V. Rancurel. Merging semantics for conflict updates in geo-distributed file systems. In *ACM Int. Systems and Storage Conf. (Systor)*, pp. 10.1–10.12, Haifa, Israel, May 2015.

[81] D. B. Terry, V. Prabhakaran, R. Kotla, et al. Consistency-based service level agreements for cloud storage. In *Symp. on Op. Sys. Principles (SOSP)*, pp. 309–324, Farminton, PA, USA, 2013.

[82] The Apache Software Foundation. Hadoop. http://hadoop.apache.org. Retrieved Oct. 2015.

[83] The Apache Software Foundation. Cassandra. http://hadoop.apache.org/, Retrieved Oct. 2015.

[84] A. Thomson and D. J. Abadi. Calvinfs: Consistent wan replication and scalable metadata management for distributed file systems. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pp. 1–14, Santa Clara, CA, Feb. 2015.

[85] A. Z. Tomsic, T. Crain, and M. Shapiro. An empirical perspective on causal consistency. In *W. on Principles and Practice of Consistency for Distr. Data (PaPoC)*, co-located with EuroSys 2015, Bordeaux, France, Apr. 2015.

[86] A. Traeger, E. Zadok, N. Joukov, et al. A nine year study of file system and storage benchmarking. *Trans. Storage*, 4(2): 5:1–5:56, May 2008. ISSN 1553-3077.

[87] A. Traeger, E. Zadok, E. L. Miller, et al. Findings from the first annual storage and file systems benchmarking workshop. *;Login:*, 33(5):113–117, Oct. 2008.

[88] J. Valerio, P. Sutra, E. Rivière, et al. Evaluating the price of consistency in distributed file storage services. In *Distributed Applications and Interoperable Systems - 13th IFIP WG 6.1 International Conference, DAIS 2013, Held as Part of the 8th International Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, June 3-5, 2013. Proceedings*, pp. 141–154, 2013.

[89] J. Valerio, P. Sutra, E. Rivière, et al. Evaluating the price of consistency in distributed file storage services. In *Distributed Applications and Interoperable Systems*, volume 7891 of *Lecture Notes in Comp. Sc.*, pp. 141–154. Springer-Verlag, 2013. ISBN 978-3-642-38540-7.

[90] R. Van Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, 1996.

[91] P. Viotti and M. Vukolić. Consistency in non-transactional distributed storage systems. ArXiv e-print 1512.00168, arXiv.org, Dec. 2015.

[92] H. Wada, A. Fekete, L. Zhao, et al. Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In *Biennial Conf. on Innovative DataSystems Research (CIDR)*, volume 11, pp. 134–143, 2011.

[93] M. Zawirski, N. Preguiça, S. Duarte, et al. Write fast, read in the past: Causal consistency for client-side applications. In *Int. Conf. on Middleware (MIDDLEWARE)*, pp. 75–87, Vancouver, BC, Canada, Dec. 2015.