



Cloud-TM

Specific Targeted Research Project (STReP)

Contract no. 257784

D1.1: User Requirements Report

Date of preparation: 30 November 2010

Start date of project: 1 June 2010

Duration: 36 Months

Contributors

Emanuel Bernard, Red Hat
Joao Cachopo, INESC-ID
Bruno Ciciani, CINI
Francesca Giannone, ALGORITHMICA
Mark Little, Red Hat
Francesco Quaglia, CINI
Luis Rodrigues, INESC-ID
Paolo Romano, INESC-ID
Vittorio A. Ziparo, ALGORITHMICA



(C) 2010 Cloud-TM Consortium. Some rights reserved.
This work is licensed under the Attribution-NonCommercial-NoDerivs 3.0 Creative Commons License. See <http://creativecommons.org/licenses/by-nc-nd/3.0/legalcode> for details.

Table of Contents

1 Introduction	3
1.1 Relationship With Other Deliverables	4
2 Requirements of the Cloud-TM Platform.....	5
2.1 The Cloud-TM Data Platform.....	5
2.1.1 Object-orientation.....	5
2.1.2 Data Querying.....	7
2.1.3 Transactional Data Manipulation.....	8
2.1.4 Persistence.....	9
2.1.5 Data Distribution and Replication.....	10
2.1.6 APIs for parallel and distributed programming.....	11
2.2 The Cloud TM Autonomic Manager.....	12
2.2.1 QoS API.....	13
2.2.2 Resource usage monitoring.....	14
2.2.3 Workload pattern tracking.....	14
2.2.4 Resource demand and workload phase prediction.....	15
2.2.5 Elastic scaling	16
2.2.6 Dynamic replication strategies.....	16
2.2.7 Dynamic contention management and data distribution schemes.....	17
3 Use Cases	19
3.1 The Pilots.....	19
3.1.1 AI-Colony.....	19
3.1.2 GeoGraph.....	20

3.2 The Cloud-TM programming paradigm.....	20
3.2.1 Asynchronous Computation of Statistical Information.....	21
3.2.2 Users interactions in the AI-Colony gameplay.....	23
3.2.3 Transactional manipulation of in-memory vs persistent data.....	26
3.3 QoS management, fault-tolerance and self-optimization.....	29
3.3.1 Quality of Service Specification	32
3.3.2 (Self-)Optimization	34
3.3.3 Disaster Recovery.....	38
4 References.....	39

1 Introduction

In the Cloud, resources are dispensed “elastically”, providing application developers with a seemingly unbounded amount of computational power and storage on a usage-based pricing model.

Unfortunately, designing and implementing software services that are actually able to match the scalability potentialities of large scale, shared-nothing Cloud infrastructures is far from being a trivial task. In fact, one of the main challenges that needs to be faced to bring about the potential of Cloud computing, and ultimately consolidate its business model, is to develop programming models and tools capable of simplify the design, implementation and administration of applications for the Cloud.

The Cloud-TM middleware platform will pursue exactly this goal by addressing the following key challenges:

- offering a simple and intuitive programming model for the implementation of services deployed in Cloud computing platforms. Specifically, by letting service developers focus on delivering differentiating business value, instead of managing low-level, error prone mechanisms (such as inter-process synchronization, data distribution, persistence and fault-tolerance), Cloud-TM will allow a major reduction of the costs associated with the development process.
- minimizing the monitoring and administration costs/efforts by automating the provisioning of resources from the Cloud, with the twofold objective of meeting user-specified Quality of Service and allowing operational costs to be consistent with predetermined budgets.
- maximizing the scalability and efficiency of the user-level services (thus improving the cost/benefit ratio in the Cloud Computing usage-based pricing model) via autonomic paradigms providing self-tuning capabilities aimed at optimizing the platform's performance in face of fluctuations of the number of allocated resources and of the workload characteristics.

In this document the Consortium's partners specify (and prioritize) the key functional requirements that will be supported by the Cloud-TM middleware platform, illustrating how they will be leveraged by Cloud applications via use-case diagrams.

The keywords used to indicate requirement levels are compliant to the best practices defined in RFC2119 by the Internet Engineering Task Force (<http://www.ietf.org/rfc/rfc2119.txt>) .

1.1 Relationship With Other Deliverables

The requirements specified in this document will serve as a reference for the majority of the future technical deliverables of the project. In particular, this document will strongly influence the definition of deliverable D2.1 “Architecture Specification Draft” and D2.2 “Preliminary prototype of the RDSTM and RSS”, which will be developed during the next 6 months.

In the longer term, the identified requirements will drive the development of the deliverables associated with the software prototypes of the Cloud-TM platform developed by WP 2 and WP 3, as well as of the pilot applications output by WP 4.

Note that it is expected that the set of requirements defined in this document will be refined based on the feedback gathered from the execution of the research and development work carried out during the rest of the project.

2 Requirements of the Cloud-TM Platform

The Cloud-TM platform will consist of two main logical modules, the Data Platform and the Autonomic Manager, whose requirements will be specified in the following.

2.1 The Cloud-TM Data Platform

The Data Platform will allow to store, retrieve and query data across a dynamic set of distributed nodes elastically acquired from an underlying Cloud infrastructure. The Data Platform will expose an API that will allow developers to access the platform's data via Plain Old Java Objects (POJOs), allowing them to fully exploit the power of object oriented programming paradigm (including inheritance, polymorphism, relations such as composition and aggregation) while sparing them from dealing with low-level issues such as distribution, fault-tolerance, concurrency and persistence.

This component will rely on the transaction abstraction as a first-class programming construct to synchronize concurrent accesses to shared data structures, leveraging on a (Distributed) Transactional Memory layer to detect conflicting accesses directly in-memory, and not relying on the concurrency control mechanisms implemented at the persistence storage layer (as in classical DBMS-based solutions).

The Data Platform will integrate a library of alternative data consistency protocols, designed to achieve optimal performance in presence of diverse workload settings (write vs read intensive, low vs high conflict rate) and different scales of the data platform.

Further, the Data Platform will expose a set of APIs for the development of parallel and distributed applications that will provide synchronization (such as barriers, conditional waits) and scheduling (e.g. queues or futures) primitives for coordinating the execution of threads physically running on different shared-nothing nodes.

2.1.1 Object-orientation

R1 The Data Platform API MUST allow to store to and retrieve from the Cloud-TM platform plain Java objects. Specifically, the Cloud-TM platform MUST maintain:

R1.1 JDK primitive types (e.g. integers, reals)

R1.2 Instances of user-defined classes

R1.3 Fully polymorphic Java objects (use of subclasses is a natural need in a OO system)

R2 The Data Platform API MUST allow to maintain object associations between entities and value objects (roughly composition in the OO world). Entities are persistent objects with their own lifecycle. The lifecycle of a value object is bound to its owning entity.

R3 The Data Platform API MUST allow to maintain object associations between entities and for all cardinalities.

R3.1 “one to one”, “many to one” are typically represented as properties in the object world.

R3.2 “one to many” and “many to many” are typically represented as a collection property in the object world.

R4 The Data Platform API SHOULD offer also support a simpler key-value storage API, such as the one specified by JCache. This will allow to more efficiently use cases that encompass caching or storage of temporary / heavily computed data.

R5 The Data Platform API MUST ensure object identity for a given unit of work, since object identity is expected from many users in an object oriented world. Having a way to keep this contract even with objects stored in the Cloud-TM platform is necessary.

R6 The Data Platform API, and, more in general the Cloud-TM platform, SHOULD NOT technically constrain (or, more pragmatically, as few as possible) the object model. The object model is at the core of modern applications and is used way beyond the data layer, such as the business layer or the presentation layer, and can be serialized and deserialized on a different VM on the client side. The more technical constraints are imposed, the more complicated it gets for adoption.

R7 The Data Platform API, and, more in general, the Cloud-TM platform, MUST work well with the rest of the PaaS infrastructure. Java EE 6 is the targeted foundation block for the PaaS infrastructure. The Data Platform API MUST interoperate easily with the rest of the platform (i.e. injection of the unit of work via CDI, declarative transactions, integrates with JTA etc).

R8 The Data Platform API MUST allow to uniquely identify an object across the Cloud-TM platform also in the common case in which this spans across a set of distributed nodes.

R9 The Data Platform API MUST allow to create, read, update, delete the objects stored/to be stored by the Cloud-TM platform. Note that such API can be more or less transparent.

R9.1 The Data Platform API to create, read, update, delete objects SHOULD be as natural as possible to a person familiar with object state management (as opposed to Data

Manipulation Language operations as seen in SQL).

2.1.2 Data Querying

R10 The Data Platform API MUST offer basic querying capabilities:

R10.1 Query by property values

R10.2 Query by range of values

R10.3 Query based on associated entity states (at least for “to one” associations).

R10.4 Query for exact values or approximative values (as in SQL like operator).

R10.5 Polymorphic queries (i.e., “select * from *User*” should return “customers” if *Customer* extends *User*).

R11 The Data Platform API SHOULD allow querying on associated entities for “to many” associations

R12 The Data Platform API SHOULD offer aggregation functions in queries, similar in fashion to the SQL aggregation functions: average, sum, min, max

R13 The querying engine of the Cloud-TM platform MUST explore various query strategies, such as distributed indexes, queries partly executed on each node (on a subset of the data set) and subsequent client aggregation, dedicated query nodes.

R14 The Data Platform API MUST allow to define the fetching profile of the object graph

R15 The Data Platform API SHOULD be familiar with the Java developer community and foster adoption. The JPA API is a good candidate for such approach both for the CRUD API and for the JP-QL (and Criteria API) approach.

R16 The Data Platform API should be easy and intuitive to use (easy packaging), to foster user adoption.

R17 The Cloud-TM platform's middleware and its APIs must be well documented and the documentation must be understood by “mere mortals”, not only by specialists of distributed and concurrent programming.

2.1.3 Transactional Data Manipulation

R18 The Data Platform API MUST allow programmers to demarcate a group of operations on the application's state into transactions.

R18.1 Transactions MUST be atomic, in the sense that, upon commit/abort of a transaction, all/nothing of the data alterations carried out within the transaction must be applied.

R18.2 Transactions MUST be isolated, in the sense that they should observe consistent snapshots of the application's data even in case this is concurrently accessed and manipulated. Note that several definitions of “consistency” have been proposed in the literature of DBMS and STM systems, and several isolation levels have been accordingly defined in international standardization bodies (e.g. ANSI/ISO SQL Isolation Levels).

R18.2.1 The Data Platform API allow programmers to define the isolation level to be enforced by the Cloud-TM platform.

R18.2.2 The Data Platform API SHOULD allow programmers to define dynamically, and on a per-transaction basis, the isolation level to be enforced by the Cloud-TM platform.

R18.2.3 The Cloud-TM platform SHOULD integrate a library of different concurrency control mechanisms to detect conflicts among transactions simultaneously executed in the same node (e.g. encounter vs commit time locking algorithms, visible vs invisible reads, write buffering vs undo logging, fine vs coarse conflict detection granularity) and allow the application developers and the Autonomic Manager to select (ideally at run-time) the scheme to use.

R18.3 The Data Platform API MUST allow programmers to define if the data accessed within a transaction should be durably persisted or not. Unlike in classic ACID transactions, in the Cloud-TM platform durability will be considered as an optional attribute for transactions. This will make the Cloud-TM programming model attractive also in parallel applications which rely on transactions to synchronize concurrent access to in-memory data structures.

R18.3.1 The Data Platform API SHOULD allow programmers to specify which subset of the data globally accessed within a transaction should be persisted upon commit of the transaction, allowing the manipulation within the same transaction of both persistent and in-memory data.

R19. The Data Platform API **MUST** support the beginning of a transaction within an already ongoing transaction (transaction nesting). This is essential to achieve composability of object-oriented programming relying on the transaction abstraction to synchronize accesses to shared data.

R19.1 The Cloud-TM platform **MUST** support flat nested transactions, where all nested transactions are added to the scope of the top level transaction. In flat nesting, the abort of a child transaction implies the abort of the top level transaction.

R19.2 The Cloud-TM platform **SHOULD** support non-flat nested transaction models, where the abort of child transactions do not cause the abort of the father transaction.

R19.3 The Data Platform API and the Cloud-TM platform **SHOULD** support parallel (non-flat) nesting of transactions. This would allow simplifying the parallelization of algorithms whose blocks can be executed in parallel provided that no data races occur among the concurrently executed portions of code (e.g. complex, long running queries).

2.1.4 Persistence

R20 The Data Platform API **MUST** allow to store the application's data into alternative persistent storages. These will encompass both locally available file-systems or databases, as well as distributed, Cloud-oriented storage services (e.g. Amazon's S3, Google's BigTable, Hadoop's Hbase, Cassandra).

R21 To maximize ease of portability towards heterogeneous persistent storages and Cloud providers, the Cloud-TM platform **MUST** adopt a modular, plug-in-based architecture. The issues associated with the interaction with diverse persistent storage systems will be encapsulated within plug-in modules which will expose the same interface towards the Cloud-TM platform.

R21.1 The Cloud-TM platform **MUST** implement plug-ins for at least two different persistent storage systems, preferably one storing data on the local hard disk drive (e.g. on filesystem) and one on a distributed persistent storage (e.g. Amazon's S3)

R22 The Cloud-TM platform **SHOULD** offer proper abstraction between the persistence schema and the object model schema, to facilitate dealing with object model evolution and allowing reading from persistent storage previous versions of data without having to convert the persisted model to the new schema.

2.1.5 Data Distribution and Replication

R23 The Cloud-TM platform **MUST** support distribution of data across a variable number of nodes dynamically allocated in an elastic fashion from the underlying Cloud infrastructure:

R23.1 The Cloud-TM platform **MUST** support efficient identification of the nodes where each object is maintained.

R23.2 The Cloud-TM platform **MUST** transparently relocate the data among the nodes upon allocation/release of nodes from/to the Cloud infrastructure, using techniques aimed at minimizing the overhead associated with data relocation (e.g. consistent hashing schemes).

R24 The Data Platform API **MUST** allow the developers to transparently access (read/write) distributed objects maintained by nodes different from the one on top of which the application's code is executing.

R25 The Data Platform API **SHOULD** allow to control on which nodes the various subsets of the application's data are stored. This API could be exploited both by the developers to specify a particular data distribution policy (e.g. aimed at co-locating data items that are typically jointly accessed by the application's logic) or by the Autonomic Manager to define self-optimizing data distribution policies and maximize applications' data access locality.

R26 The Data Platform API **SHOULD** allow to control on which nodes the various parts of the application's logic (e.g. object's methods) should be executed. This will allow to enhance applications' performance by consenting to co-locate code and data.

R27 The Cloud-TM Platform **MUST** support in-memory replication of application's data to avoid data losses and ensure data availability in the case of crashes of individual nodes.

R28 The Data Platform API **MUST** allow to control the degree of replication of the data stored by the Cloud-TM platform, namely the number of nodes on which the data items are replicated.

R28.1 The Cloud-TM platform **SHOULD** allow to dynamically vary the degree of data replication.

R28.2 The Data Platform API **MIGHT** allow to specify different degrees of replication for distinct data sets.

R29 The Cloud-TM Platform **SHOULD** integrate caching mechanism schemes aimed at reducing the frequency of accesses to remote nodes to fetch popular data items. Unlike the replication mechanisms mentioned in requirements R7.5 and R7.6, where data is consistently replicated and data updates are propagated across the various replicas, the caching mechanism will rely on more

lightweight (and relaxed) consistency mechanisms based on lazy or epidemic invalidation schemes.

R30 The Cloud-TM platform MUST implement a set of alternative protocols to ensure consistency in presence of distributed and replicated data. These SHOULD include single vs multi-master replication mechanisms, full vs partial replication protocols, eager vs lazy locking strategies, two phase commit-based vs total order-based distributed coordination schemes.

R30.1 The Cloud-TM platform MUST support time-interleaved coexistence of replication policies based on differentiated paradigms (primary-backup vs active replication). This will allow the application developers and the Autonomic Manager to dynamically configure/adapt the platform's consistency mechanisms depending on the current workload and scale of the platform.

2.1.6 APIs for parallel and distributed programming

R31 The Cloud-TM platform MUST provide APIs to synchronize the execution of threads across the Data Platform, such as barriers, count down latches, join conditions, conditional waits.

R32 The Cloud-TM platform SHOULD provide APIs for scheduling the execution of threads running across the Data Platform, such as queueing and distribution of tasks, asynchronous and periodic task execution.

R33 To make the API familiar to Java programmers, the Cloud-TM platform SHOULD expose an API as similar as possible to the Java 5 Concurrency API

2.2 The Cloud TM Autonomic Manager

The Autonomic Manager will automate the elastic scaling of the Data Platform and orchestrate the self-optimizing strategies that will dynamically reconfigure the data distribution and replication mechanisms to maximize efficiency in dynamic workload scenarios.

This component will expose an API allowing the specification and negotiation of applications' QoS requirements and budget constraints. This information will be used by the Autonomic Manager to define both the objective and the constraint functions on whose basis the elastic scaling and self-tuning policies of the Cloud-TM platform will be derived.

The Autonomic Manager will leverage on pervasive monitoring mechanisms that will not only track the utilization of heterogeneous system-level resources (such as CPU, memory, network, disk utilization), but also characterize the workload of the various subcomponents of the transactional Data Platform (local concurrency control, data replication and distribution mechanisms, data contention level) and their efficiency.

Self-optimization capabilities within Cloud-TM are related to both reactive and proactive reconfiguration approaches that rely on hybrid analytical/machine learning-based models, sophisticated mechanisms to anticipate resource demand and workload patterns and provide timely switches to optimized scale/configuration/costs.

Based on this information, the elastic scaling mechanisms provided by the Autonomic Manager will minimize the number of resources allocated from the Cloud infrastructure (and thus the operational cost for the Cloud application). The auto-scaling mechanisms will be designed to leverage on the emerging standard APIs for resource provisioning from Cloud infrastructures, maximizing interoperability in scenarios characterized by heterogeneous providers and supporting seamless migration from private to public clouds.

In parallel, the Autonomic Manager will self-optimize the mechanisms used across the various layers of the platform (local contention management, data replication and distribution mechanisms, group communication system parameters) to ensure maximal efficiency and adequate failure resiliency levels in presence of dynamic workloads and fluctuations of the scale of the Data Platform (due to failures or intentional deallocation of resources).

2.2.1 QoS API

R34 The Autonomic Manager MUST provide an API (called QoS API) to allow users to define the desired QoS levels for their applications and the maximum sustainable operational costs

R35 The QoS requirements expressed via the QoS API MUST encompass both performance and reliability metrics

R35.1 The QoS requirements expressed via the QoS API MUST be able to define desired levels of utilization of physical resources (e.g. avg. CPU utilization < 70% in every node) .

R35.2 The QoS requirements expressed via the QoS API MUST be able to define desired performance metrics of the Data Platform, such as average transaction execution time or transaction abort rate.

R35.2.1 Such performance metrics MAY be specialized for different classes of transactions. This would allow to define different QoS requirements for transactions having heterogeneous resource demands (e.g. long running vs short transactions)

R35.3 The QoS requirements targeting performance metrics, such as transaction execution time, MUST be expressible in terms of average values (whenever this makes sense).

R35.3.1 The QoS requirements targeting performance metrics, such as transaction execution time, MAY be expressible using higher order moments (e.g. variance) or percentiles (e.g. 95% of transaction execution time is within an acceptable threshold).

R35.4 The specification of the reliability-oriented QoS parameters MUST include parameters such as, the degree of replication of the platform's data and whether synchronous/asynchronous persistence is required.

R35.4.1 The specification of the reliability-oriented QoS parameters MAY allow defining additional constraints on the physical location of the nodes maintaining replicas of the same data (e.g. in the same data-center/availability zone or not).

R35.4.2 The specification of the reliability-oriented QoS parameters MAY allow defining desirable availability levels (e.g. in terms of number of down-time hours per year) for the whole platform. These metrics would be used by the Autonomic Manager to automatically define the required strategies/parameters for data replication/distribution/persistence.

R36 The QoS API SHOULD be based on emerging standards (such as OVF and the WS-

Agreement framework) for the specification and negotiation of SLAs in Cloud environments to maximize portability.

R37 THE QoS API SHOULD allow applications to dynamically alter and renegotiate their SLA parameters.

2.2.2 Resource usage monitoring

R38 The Autonomic Manager MUST monitor the usage of CPU, RAM, storage systems and communication devices.

R39 The Autonomic Manager MUST monitor resource utilization by relying on standard monitoring APIs/applications provided by basic infrastructural layers (e.g. the “top” application on POSIX compliant operating systems), or by de-facto reference API/services for Cloud Environments (e.g., the “CloudWatch” service specified by AWS). This will maximize portability and allow to benefit from any future updates of these layers/services in a transparent fashion for the user applications (provided that the associated APIs do not vary).

R40 The Autonomic Manager MUST implement monitoring subsystems providing their output in real-time on the basis of on-line gathered information.

2.2.3 Workload pattern tracking

R41 The Autonomic Manager MUST generate a workload characterization of the data access pattern exhibited by the applications layered on top of the Data Platform:

R41.1 The Autonomic Manager MUST gather statistical information, possibly at differentiated granularity levels, on the utilization and contention of application data, as well as on its size.

R41.2 The data access pattern characterization SHOULD be based on profiling information gathered at the granularity of a single transaction.

R41.3 The data access pattern characterization MAY be based on profiling information gathered at the granularity of a single access to transactional objects. Such a fine grain tracing and characterization of the activities within transactions would allow to identify the data requirements of specific phases of a transaction.

R42 The Autonomic Manager MUST generate a workload characterization of the replica

consistency management protocols.

R43 The Autonomic Manager **MUST** generate a workload characterization of the group communication protocols

R44 The Autonomic Manager **SHOULD** correlate resource usage across the different layers of the Data Platform.

R45 The Autonomic Manager **SHOULD** profile protocols and activities within the Cloud-TM platform by relying, as much as possible, on the capabilities already offered by software layers/technologies that will be ultimately selected for the corresponding implementations to minimize intrusiveness.

R45.1 The profiling subsystems **MUST** operate in real-time on the basis of information gathered on-line.

R45.2 Profiling of protocols and activities within the Cloud-TM platform **MAY** be carried out via ad-hoc software probes in order to complement other profiling services

R45.3. The profiling subsystems **MAY** be based on a mixture of information gathered on-line and knowledge bases possibly built by (i) exploiting early deploy, tracing and verification of pilot applications, and (ii) off-line benchmarking and assessment of the employed replication and group communication protocols.

2.2.4 Resource demand and workload phase prediction

R46 The Autonomic Manager **MUST** implement resource demand and workload prediction methods operating in real-time.

R47 The Autonomic Manager **SHOULD** implement prediction methods based on temporal correlation between resource usage samples and workload pattern samples.

R48 The Autonomic Manager **SHOULD** employ methods to predict performance and operational costs of the Cloud-TM platform based on workload models.

R49 The Autonomic Manager **SHOULD** implement workload prediction methods based on synthetic workloads built offline providing models derived from workload pattern tracking.

R50 The Autonomic Manager **MAY** dynamically adjust off-line built workload models (e.g. in terms of the shape of the distribution of specific parameters describing the workload) based on the on-line tracking and analysis of workload patterns.

R51 The Autonomic Manager MAY implement cost/performance prediction methods operating according to different prediction windows (e.g. hourly or less, daily, weekly, monthly, etc.)

2.2.5 Elastic scaling

R52 The Autonomic Manager MUST dynamically scale up/down the amount of resources provisioned from the underlying Cloud infrastructure, in order to ensure the QoS requirements specified by the application at minimal operational costs. This may encompass:

- dynamically increasing/decreasing the number of virtualized nodes acquired from the Cloud platform;
- dynamically increasing/decreasing the computational power (CPU speed or number of cores) and the amount of RAM available in each virtualized node (provided that the underlying Cloud infrastructure supports this functionality);
- dynamically increasing/decreasing the amount of stable storage systems allocated from the underlying Cloud infrastructure.

R53 The Autonomic Manager SHOULD dynamically scale up/down the amount of software components in execution within the Cloud-TM platform. This can encompass both instances of the Data Platform middleware and other external middleware components such as J2EE application servers.

R54 The elastic scaling mechanisms MUST take into account the notion of Availability Zones, each of which identifies a confined system region where the occurrence of faults is not correlated with the occurrence of faults in other regions. This will allow supporting the dynamic re-definition of the level of coupling among the resources in order to confine (possible) failures within disjoint resource subsets.

R55 The elastic scaling mechanisms MUST operate in real-time.

R56 The elastic scaling mechanisms SHOULD rely on emerging standard, vendor independent APIs for resource provisioning in public Cloud providers (e.g. AWS), to maximize interoperability in multi-provider scenarios.

R57 The Autonomic Manager SHOULD implement autoscaling policies spanning across differentiated/heterogeneous Cloud provisioning systems or Cloud providers.

2.2.6 Dynamic replication strategies

R58 The Autonomic Manager MUST dynamically select the replication strategy that maximizes the

performance of the Cloud-TM platform givent the current workload.

R58.1 The choice of the replication strategy SHOULD explicitly take into account all the cost factors at the level of the underlying virtualized infrastructure. This aspect is related to the fact that typical cloud provisioning services expose both fixed cost resources (per time unit) and variable cost resources. Data transfer falls in the latter class depending on the deploy of the end-points, and the actual data transfer volume (over time) has relations with the used coordination protocol across replicated components, depending on the replication strategy.

R59 The Autonomic Manager SHOULD dynamically rely on persistent storage facilities whenever the failure resiliency achieved via in-memory replication falls under user-defined thresholds (e.g. due to loss of connectivity with some or all the remaining replicas).

R60 The Autonomic Manager MUST support dynamic replication switching policies based on heuristic methods.

R61 The Autonomic Manager SHOULD support dynamic replication switching policies based on analytical, statistical or simulation models capturing both performance and cost aspects.

R62 The Autonomic Manager SHOULD support dynamic replication switching policies based on machine learning techniques (e.g. Reinforcement Learning).

2.2.7 Dynamic contention management and data distribution schemes

R63 The Autonomic Manager SHOULD dynamically switch to the contention management strategy that provides the most appropriate level of overlapping between local computation and replica coordination. This requirement entails the ability to adapt the system behavior according to cross-layer optimizations triggered by variations of the operating mode of the group communication layer coordinating replicas' activities. This, in turn, is related to dynamic variations of the size/deploy of the virtualized infrastructure (e.g. the current size/deploy may impact spontaneous ordering guarantees of replica broadcasted data).

R64 The Autonomic Manager MAY rely on off-line application analysis techniques aimed at determining the actual level of isolation required by the application to achieve serializability (e.g. for specific application patterns, a concurrency control mechanism ensuring only snapshot isolation is sufficient to filter out any non-serializable schedule). This requirement is related to the aforementioned issue concerning the optimization of infrastructural costs since lower isolation level

may require replica coordination schemes imposing reduced costs in relation to, e.g., data transfer volumes.

R65 The Autonomic Manager **MUST** support remapping of application data, e.g. upon failure or unreachability of nodes maintaining copies of data.

R65.1 This re-mapping **SHOULD** also take into account the possible correlation of failures among the nodes selected for maintaining copies of the data (e.g. selecting nodes belonging to different Availability Zones in order to guarantee or restore predetermined failure resiliency levels).

R66 The Autonomic Manager **SHOULD** support dynamic remapping of (replicated) data copies in order to optimize data locality and client proximity depending on the current (or predicted) workload pattern and the current virtualized infrastructure deploy.

R67 The Autonomic Manager **SHOULD** support dynamic remapping of data by relying on reward models explicitly taking into account the economic cost for the remap operation. This requirement is related to the aforementioned infrastructural aspect on data transfer pricing models proper of Cloud environments.

3 Use Cases

In this section, we provide a series of case studies for the Cloud-TM platform that are based on the two pilot applications AI-Colony and GeoGraph. AI-Colony is a particular class of Massively Multiplayer Online Game called office games, while GeoGraph is a framework for the development of Location-based Mobile Social Networking (LMSN). The case studies provide concrete examples that motivate the main requirements of the Cloud-TM platform discussed in Section 2.

In the following, we start by providing a brief overview of the pilot applications (Section 2.1), and then we detail the corresponding use cases.

First, we describe the use cases relative to the development phase (Section 2.2) of applications, where we show how the Cloud-TM's transactional functionalities can be leveraged to simplify the development of the pilot applications.

Next, we describe the use cases associated with the QoS management, fault-tolerance and self-optimization features (Section 2.3) of the Cloud-TM platform, illustrating how they will allow to reduce the operational costs of applications deployed in Cloud computing infrastructures.

3.1 The Pilots

3.1.1 AI-Colony

Office games are Massively Multiplayer Online Games allowing for an occasional playing style (e.g. during coffee breaks at the office, by which its name is suggested), but, nevertheless, awarding the most assiduous players. AI-Colony is an innovative office game capable of addressing the need for flexibility of office game users. In AI-Colony, users lead small colonies of software agents that inhabit a virtual world.

The key idea is that users are required to control their colony through high level commands, while agents are capable of handling user offline periods performing ordinary and time-consuming tasks, such as territory defense and production. As almost any office-game, the business model underlying AI-Colony prescribes basic game-play for free and requires users to pay for extra features. The main difference with respect to other games, is that the extra features demand computation power as they may encompass increasing the number of agents and the the complexity of their behavioral algorithms to enhance the colony performance during user off-line

phases.

3.1.2 GeoGraph

Location-based Mobile Social Networking (LMSN), also known as geo-social networks are a second generation Social Networks that can take advantage of the position of users in order to provide innovative services. A typical LMSN application is a system built upon a client-server architectures, where:

1. Clients are apps on SmartPhones that periodically send location data and that can retrieve information on friends that are close;
2. Server-side applications are in charge of maintaining a consistent representation of the data and are required to provide up to date information on the relations among users based on their location. For example, a location mobile social network may require to maintain a data structure that dynamically keeps track of all the friends of each user within a given distance.

GeoGraph will be an open-source framework for tracking location data and dynamically computing relationships among users. In particular, GeoGraph will compute and dynamically update a graph (called GeoGraph) where nodes are users and where edges connect users who are within a given distance from each other. Being a generic framework, GeoGraph will support the development of a wide range of LMSN applications characterized by highly heterogeneous and dynamic workloads exhibiting diverse data access patterns and (transaction) conflict rates. For the purpose of this project the GeoGraph pilot will be developed as a tool intended for aiding the development of LMSN server side applications and evaluating the Cloud-TM platform. Thus, we shall not implement any Smartphone client, but rather provide a user simulator in order to generate diverse workloads profiles.

3.2 The Cloud-TM programming paradigm

When dealing with massive numbers of embodied agents acting on a shared virtual environment, concurrency issues become very compelling. Explicit programming of concurrency is extremely time-consuming and risk prone, yielding to prohibitive development and testing costs. The transactional API available within the Cloud-TM platform offers an elegant solution to this problem, allowing for a faster and more reliable development process.

While consistency requirements for LMSN applications are typically “weak” ones, since

inconsistent states are just transitory, consistency requirements for MMOGs are indeed “strong”. Roughly, one can define a game as a set of moves available to players and a set of rules that define when moves are applicable and their effects. When moves are executed concurrently, in order to enforce consistency, the verification of the applicability of a move and the execution of the move itself require to be within a (read/write) transaction. For example, say that there is a move “transport(object, source, destination)” that allows to move an object from a source location to a destination and that it is applicable when the object is at the source location. Without transactional support, it could happen that two players concurrently verify that a transport move is applicable and, then, that both players execute the transport move with two different target locations. This would yield to the creation of two different copies of the original object, leading the system to an inconsistent state with a significant impact on the game-play.

In the following, we describe use cases relative to the transactional features available to developers of the Cloud-TM platform. The use cases address different consistency requirements related to the pilot applications.

3.2.1 Asynchronous Computation of Statistical Information

In this use-case, we address the task of collecting statistical data for GeoGraph. GeoGraph will offer a service that computes periodically and asynchronously statistical data (e.g., user density on a per-zone basis), and that will persist the results for later convenience. Strong consistency requirements are too demanding for this type of queries, as the high frequency of updates of the node positions could generate an undesirably high number of conflicts. Also, given the mobile nature of the nodes and the periodicity of the statistics computation, the statistical information could ineluctably become stale after a very short period of time. In other words, this statical information is, due to its own nature, weakly consistent.

Thus, in this scenario, the developers may opt to rely on more relaxed consistency criteria, e.g. repeatable read rather than serializability, reducing the overhead (e.g. in terms of conflict detection) for the Cloud-TM platform.

Unique id	UC321_1
Type	Use Case
Short Name	GeoGraph Stats
Description	The system, periodically and asynchronously, queries the GeoGraph for the average density of (several kinds of) users on a per-zone basis. The result of the query, is persisted and is available for inspection by the administration.
Related Requirements	<ul style="list-style-type: none"> • Programming R1 to R9 • Queries R10 to R17 • Transactions R18 • Persistence R20 to R22
Involved Stakeholder	Developer
Involved Cloud-TM Components	Data Platform, Distributed STM
Involved Pilot	GeoGraph

Table 1: GeoGraph Stats

3.2.2 Users interactions in the AI-Colony gameplay

This use case is aimed at addressing the strong consistency requirements of AI-Colony. There are two main activities that are executed concurrently in the game:

- foraging depletable resources (Table 2)
- building infrastructures (Table 3)

All these activities have strong consistency requirements that require a serializable isolation level. As an example, let us consider the activity of building a colony. Roughly, you may think of AI-Colony as an environment represented as a graph where players can build roads on edges and colonies on nodes. According to the current gameplay, the preconditions for a player "p" of building a colony on a node "n" are that:

1. the node "n" is reached by a road of "p";
2. all of the adjacent nodes (i.e., directly connected by an edge to "n") are vacant (i.e., none are occupied by any settlements).

Building a colony on a node requires a single transaction to check the adjacent nodes in order to verify that they are vacant and then, if this condition is satisfied, to add the colony to the node. Clearly, the transaction needs to read the state of the adjacent nodes but does not change it. Moreover, it is very likely that someone concurrently tries to build a settlement in an adjacent node, as races on the placement of settlements in strategic locations is one of the key aspects of the game mechanics. In particular, a big part of the gameplay consists in trying to place settlements in "rich areas" to exploit them and to avoid the expansion of other players.

Unique id	UC322_1
Type	Use Case
Short Name	AI-Colony Foraging
Description	In the environment there are locations that contain finite and discrete amount of resources that are generated periodically. Players, through their colonies, concurrently access these locations and forage small amounts of resources until the resources are depleted.
Related Requirements	<ul style="list-style-type: none"> • Programming R1 to R9 • Queries R10 to R17 • Transactions R18 • Persistence R20 to R22
Involved Stakeholder	Developer
Involved Cloud-TM Components	STM
Involved Pilot	AI-Colony

Table 2: AI-Colony Foraging

Unique id	UC322_2
Type	Use Case
Short Name	AI-Colony Building
Description	<p>Players concurrently build colonies and roads in the environment. These infrastructures must be built only if the target location is free. Moreover, the applicability of these actions depends on the state of the environment locally to the target location.</p> <ul style="list-style-type: none"> • Roads can be built by a player only if connected to one of its colonies or to one of its road. • Colonies must be connected to a player's road, but there also must not be any other colony in the neighborhood.
Related Requirements	<ul style="list-style-type: none"> • Programming R1 to R9 • Queries R10 to R17 • Transactions R18 • Persistence R20 to R22
Involved Stakeholder	Developer
Involved Cloud-TM Components	Data Platform, Distributed STM
Involved Pilot	AI-Colony

Table 3: AI-Colony Building

3.2.3 Transactional manipulation of in-memory vs persistent data

Depending on the application domain, it may be required to persist some portions of data while, for other portions, it may be required to just store them in memory (possibly replicating them across multiple nodes to enhance data availability and failure resiliency). For example,

- GeoGraph's geo-social graph is planned to be maintained in memory. In fact, the information on the position of users stored in the nodes of the geo-social graph is updated with a high frequency and is provided automatically by clients as the users log in. In a similar way, the topology of the geo-social graph is very dynamic and requires continuous updates as the users move. In this case, the overhead required in persisting the graph is not acceptable, as losing the geo-social graph data is not an issue. Thus, all transactions on the geo-social graph will be manipulating fully-in-memory data. Nevertheless, periodically and asynchronously, snapshots of the graph and aggregated data may be persisted, e.g., for statistical purposes.
- AI-Colony's data is planned to be persisted because it is fundamental to the users that the system can recover data after failure. The state of the system represents the achievements of players within the game and losing even the results of a few game episodes could be perceived as an unacceptable loss by the users. In this case, the overhead of persisting data could be accepted in favor of stronger guarantees on the reliability of the system.

In the following, we provide two use cases that address the cases when transactions alter only data in memory (Table 4) or only persisted data (Table 5).

Unique id	UC323_1
Type	Use Case
Short Name	GeoGraph In-Memory
Description	<p>This scenario, is aimed at testing the transactional management for two core functionalities of GeoGraph through two use cases:</p> <ul style="list-style-type: none"> – Updates to geo-graph, triggered by the position update of users – Queries on neighborhood relationships in the geo-graph
Related Requirements	<ul style="list-style-type: none"> • Programming R1 to R9 • Queries R10 to R17 • Transactions R18 • Persistence R20 to R22
Involved Stakeholder	Developer
Involved Cloud-TM Components	Data Platform, Distributed STM
Involved Pilot	GeoGraph

Table 4: GeoGraph Transactions on In-Memory Data

Unique id	UC323_2
Type	Use Case
Short Name	All-Persistent
Description	<p>Transactions alter only persisted data. This scenario, is aimed at testing the transactional management for the core functionalities of AI-Colony and GeoGraph stats through two use cases:</p> <ul style="list-style-type: none"> – Transactions related to gameplay activities in AI-Colony. – Transactions related to the persistence of GeoGraph stats and user data.
Related Requirements	<ul style="list-style-type: none"> • Programming R1 to R9 • Queries R10 to R17 • Transactions R18 • Persistence R20 to R22
Involved Stakeholder	Developer
Involved Cloud-TM Components	Data Platform, Distributed Persistent Storage
Involved Pilot	AI-Colony, GeoGraph

Table 5: Pilots and Transactions on Persistent Data

3.3 QoS management, fault-tolerance and self-optimization

The deployment and management of Cloud applications can be very challenging, as there are critical issues related to: the specification the quality of service (QoS), the system optimization required to meet the desired QoS levels and, when data-loss is a sensible issue, the recovery procedures after disasters like the failure/unreachability of a data-center.

Cloud-TM introduces a novel business model characterized by the decoupling between what is demanded to the software development company, and what is demanded to the company in charge of the in-service phase. This means that who develops the software must not design the system in face of some predicted requirements on the size of the hardware. Moreover, who is in charge of the deployment, does not need to worry about how to size the system (in terms of computational, storage, network and software resources) to ensure predefined QoS levels as the Cloud-TM platform will autonomously take care of this. The latter is a key-feature as applications such as, for example, LMSN and MMOG, may experience rapid, unpredictable, fluctuations of the user base growth rate, as testified by well known examples (e.g., FourSquares, Facebook, Gameforge AG or Zynga).

The capability of the Cloud-TM platform to elastically scale and self-optimize in real-time in order to maintain the desired QoS levels is a key factor to maximize user satisfaction and reliefs developers from the burden of designing the system based on specific and unreliable workload predictions..

Figure 2 illustrates a possible example scenario highlighting the autonomic, self-optimizing capabilities of the Cloud-TM platform. In the bottom part of the figure we show the variations over time of i) the incoming traffic, ii) the ratio of read/write transactions, and iii) the transaction conflict probability. We consider five different workload scenarios, each one associated with distinct optimal management strategies minimizing the number of resources to be hired from the cloud (while, of course, ensuring the same, user-defined Service Level Agreements).

In the first scenario (from the left) the workload is read-dominated and both the incoming traffic and the transaction conflict probability are low. In these settings, the optimal strategy is to hire a low number of resources and to use a primary-backup replication scheme [Budhiraja et al., 1993], where all the writes are processed by a single node (also called master node) that propagates the updates to the remaining ones. It is well known, in fact, that this simple, centralized approach allows minimizing the replica coordination overhead, providing optimal performances as long as the number of write transactions remains sufficiently low so to avoid overloading the primary. As the load increase (second scenario from the left), leaving unaltered the conflict rate and the write/read ratio, the Cloud-TM will automatically scale up the number of nodes hired from the Cloud

infrastructure. The replication scheme does not vary with respect to the former scenario. In fact, being the workload read-dominated, the primary is still able to timely process the incoming write transactions.

In the following scenario (the third one from the left), due to a surge in the relative number of write transactions, the primary gets overwhelmed by the incoming write requests, becoming the system's bottleneck. The Cloud-TM will react by triggering a reconfiguration of the replication scheme, passing to employ a multi-master replication scheme [Schneider, 1993], which allows for an even distribution of the write requests among the nodes and, consequently, the achievement of a higher throughput. Since the transaction conflict rate remains low (because the write operations target low contended data regions, e.g. private user data), an optimistic conflict detection policy relying on a certification-based approach [Kempe and Alonso, 1998; Rodrigues et al., 2002] is adopted. In fact, by letting transactions optimistically access possibly stale data, this class of replication protocols allows achieving maximum inter-replica parallelism, at least as long as the conflict rate remains sufficiently low [Ciciani et al., 1992].

In the fourth scenario we consider precisely the case of increase of the transaction conflict rate. This type of workload may arise, for instance, in an e-auction application in proximity of the expiration of an auction for a highly contended item. In high-contention scenarios, optimistic conflict detection schemes suffer of a high abort rate and incur in a significant performance degradation. The Cloud-TM middleware will detect such a suboptimal system configuration, and transparently switch to a more pessimistic conflict detection scheme, e.g. lease [Romano et al., 2009c] or token [Son and Kouloumbis, 1993] based, which are known to perform better in high conflict scenarios by serializing transactions and reducing the transaction abort probability [Ciciani et al., 1992; Nicola et al., 2000].

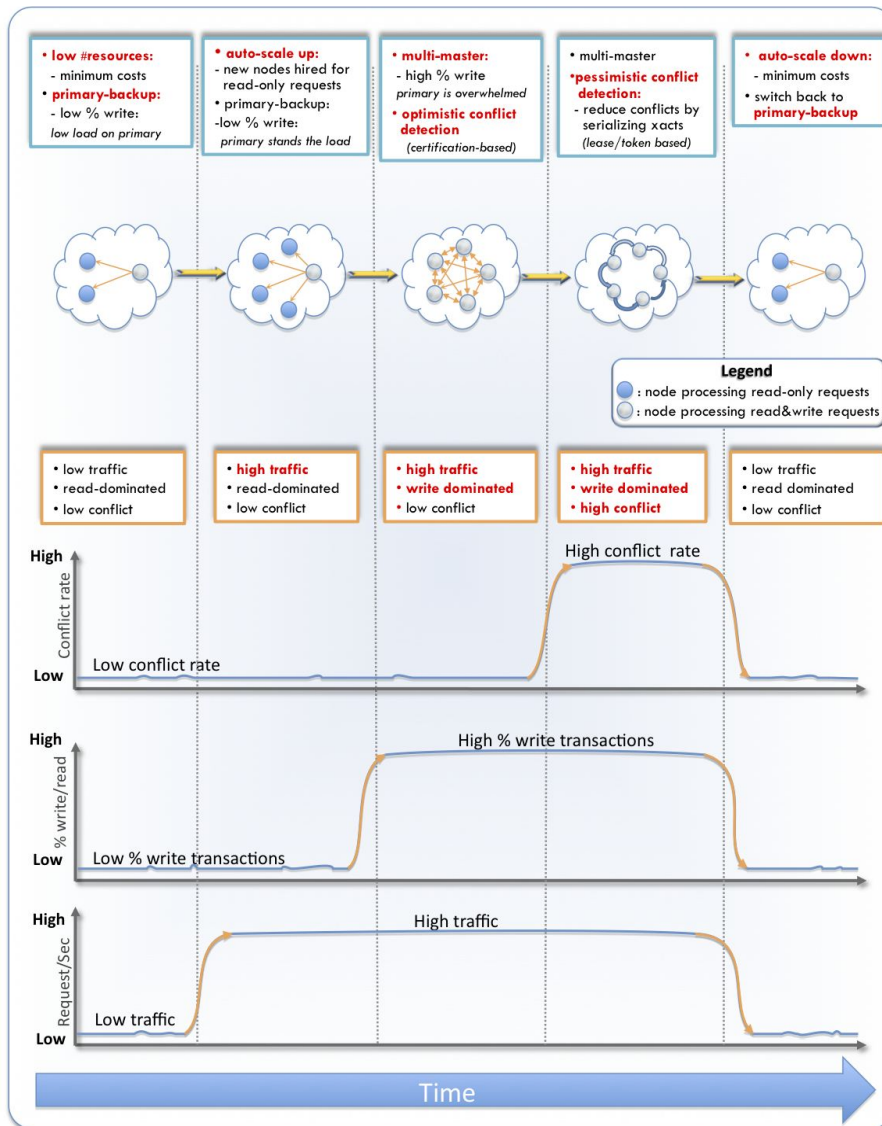


Figure 1: Example of self-optimization

Finally, to the extreme right, as the workload returns to exhibit its original characteristics (low load, write/read ratio and conflict rate), the Cloud-TM will accordingly release the additional nodes acquired from the Cloud infrastructure and switch back to use a primary-backup scheme (that outperforms multi-master approaches in these workload conditions). This will allow to avoid underutilizing the hired resources and to minimize the operational costs sustained by the Cloud user.

The use cases in the following provide possible scenarios which may be faced by the pilot applications once deployed. Since GeoGraph is a framework for building LMSN we will consider for this scenario a generic LMSN application that allows for posting geo-localized messages and for

notifying the proximity of friends, as these two features are very common in LMSN applications.

3.3.1 Quality of Service Specification

The ability to define the QoS is fundamental requirement for system administrators. Administrators must be must be able to specify the desired level of service in terms of performance and reliability, while being able to satisfy budget constraints.

The Autonomic Manager will provide an API (called QoS API) to allow users to define the desired QoS levels for their applications and the maximum sustainable operational costs. The QoS requirements expressed via the QoS API will encompass both performance and reliability metrics, and can be constrained to a budget.

Unique id	UC331
Type	Use Case
Short Name	Quality of Service
Description	<p>The administrator can set two different QoS levels for the two pilots:</p> <ul style="list-style-type: none"> • GeoGraph must ensure the processing of each position update within 2 seconds, so to ensure real-time tracking. In terms of reliability guarantees, it is acceptable to tolerate the loss of some position updates due to node failures, as the stream of updates will permit to rapidly reconstruct the most recent application's state. Budget can be adjusted as the system grows. • AI-Colony requires an average processing time for user requests inferior to 5 seconds, which is largely acceptable in video-games having a slow gameplay dynamic. On the other hand, the Cloud-TM platform should enforce strong consistency also in presence of node failures, as the loss or the corruption of the game state due to faults would represent an extremely serious issue. Budget can be adjusted as the system grows.
Related Requirements	From R34 to R37
Involved Stakeholder	Administrator
Involved Cloud-TM Components	Autonomic Manager, QoS API
Involved Pilot	AI-Colony, GeoGraph

Table 6: Quality of Service

3.3.2 (Self-)Optimization

The ability to enforce a quality of service within the Cloud-TM platform while minimizing cost means that it has to be able to adapt to varying workloads in terms of traffic, transactional conflicts and read/write ratio. As already discussed, these functionalities are provided by the Autonomic Manager of the Cloud-TM platform. The following use cases describe possible dynamics of the workload for the two pilots.

It must be noticed that while GeoGraph applications have a wider range of possible workload profiles (Table 7), in AI-Colony the workload profile appears to be more restricted because of the game mechanics (Table 8). Nevertheless, both case studies are capable of triggering the Cloud-TM mechanisms for transparently and autonomously scaling the number of computational resources, while dynamically adjusting the its configuration to the different workloads it will face.

Unique id	UC332_1
Type	Use Case
Short Name	GeoGraph Self-optimization
Description	<p>Consider the following possible evolution phases for the generic LMSN application described above, and their associated workload profiles:</p> <ol style="list-style-type: none"> 1. [low traffic, low conflicts, read-dominated] An LMSN application, when first launched, has few users (i.e., low traffic) that usually are not geographically dense. In this case, the application is used primarily as a geo-localized micro-blog as the probability of user encounters is low. For this reason, the frequency of writes (posts) is generally lower than the frequency of reads (people reading the posts). 2. [hi traffic, read dominated, low conflict] When the LMSN application becomes more well known, and the user base increases, there will be more traffic. Nevertheless, the ratio between reads and writes remains dominated by reads. 3. [hi traffic, write dominated, low conflict] As the user base grows, there may be users starting to use the LMSN for finding friends near to them. Yet the probability of meeting other users is still low, as the user base is still rather geographically sparse. Nevertheless, the system must track the position of these users and this means that their clients send their position to the server with a high frequency. In this case, also with a limited amount of users, clients can generate a high traffic on the server, which is mainly write dominated. Notice that, in this case the number of conflicts is low as the probability of being close someone (and thus add an edge to the graph) is low. 4. [hi traffic, write dominated, hi conflict] When the user base gets massive, there is a high traffic generated by the the position

	tracking system which mainly write dominated. Nevertheless, there are is high number of conflicts because of the need to maintain the topology of the GeoGraph in dense areas with many moving users, e.g., in big cities on weekends.
Related Requirements	<ul style="list-style-type: none"> • Workload Tracking: from R41 to R45 • Demand and Prediction: from R46 to R51 • Scaling: from R52 to R57 • Replication: from R58 to R62 • Contention Management and Data Distribution: from R63 to R64
Involved Stakeholder	Administrator
Involved Cloud-TM Components	Autonomic Manager
Involved Pilot	GeoGraph

Table 7: GeoGraph and Self-optimization

Unique id	UC332_2
Type	Use Case
Short Name	AI-Colony and Self-Optimization
Description	<p>Consider the following possible evolution phases of AI-Colony and their associated workload profiles:</p> <ol style="list-style-type: none"> 1. [low traffic, hi conflicts, read/write] When a office game as AI-Colony is first launched it it will probably have a small user base. These players will generate a low traffic that will be characterized by both writes and reads, as for every move of a player the system needs to read the state of the world to check if move is applicable, and if so, it needs to write to update the world state in order to apply the effects of the action. This kind of activity will generate many conflicts (e.g., by players competing on shared resources). 2. [hi traffic, hi conflict, read/write] When the user base gets massive, the type of workload will not change, although the traffic will get significantly higher.
Related Requirements	<ul style="list-style-type: none"> • Workload Tracking: from R41 to R45 • Demand and Prediction: from R46 to R51 • Scaling: from R52 to R57 • Replication: from R58 to R62 • Contention Management and Data Distribution: from R63 to R64
Involved Stakeholder	Administrator
Involved Cloud-TM Components	Autonomic Manager
Involved Pilot	AI-Colony

Table 8: AI-Colony and Self-Optimization

3.3.3 Disaster Recovery

The main asset of a company that produces applications for large masses of users, such as GeoSocial applications or MMOGs, is user data. User data provides the means for user retention and is a fundamental resource for marketing. As the user base of an application grows, the policies for recovery become more critical and replication in a single data-center becomes inadequate.

Indeed, if for a small application the expected cost of a disaster (i.e., the probability of disaster times the value of the user data) is acceptable, this may not be true in the case of applications with an extremely large user base. In this case, one would prefer to back-up data in more than one data-center, being able to restore the data in the case of the loss of the entire data in one data-center.

Unique id	UC333
Type	Use Case
Short Name	Disaster Recovery
Description	Administrators must be able to: <ul style="list-style-type: none">• specify, on a geographical scale, data-centers for replication• restore data from a source to a destination data-center
Related Requirements	Distribution Schemes: from R65 to R67
Involved Stakeholder	Administrator
Involved Cloud-TM Components	Autonomic Manager, Data Platform
Involved Pilot	AI-Colony, GeoGraph

Table 9: Disaster Recovery

4 References

- [Budhiraja et al., 1993]** Budhiraja, N., Marzullo, K., Schneider, F., and Toueg, S. (1993). The Primary-Backup Approach. In Mullender, S., editor, *Distributed Systems*, pages 199–216. ACM Press.
- [Ciciani et al., 1992]** Ciciani, B., Dias, D.M. and Yu, P.S. (1992). Analysis of concurrency-coherency control protocols for distributed transaction processing systems with regional locality. *IEEE Transactions on Software Engineering*, 18(10), pp.899–914.
- [Kemme and Alonso, 1998]** Kemme, B. and Alonso, G. (1998). A suite of database replication protocols based on group communication primitives. In *Proc. of the The 18th International Conference on Distributed Computing Systems*, page 156, Washington, DC, USA. IEEE Computer Society.
- [Nicola et al., 2000]** Nicola, M. and Jarke, M. (2000) Performance Modeling of Distributed and Replicated Databases. *IEEE Trans. on Knowl. and Data Eng.* 12, 4 (Jul. 2000), 645-672.
- [Romano et al., 2009c]** Romano, P., Rodrigues, L., and Carvalho, N. (2009). The Weak Mutual Exclusion Problem. In *Proceedings of the International Conference on Parallel and Distributed Systems (IPDPS)*, Rome, Italy, IEEE Computer Society
- [Rodrigues et al., 2002]** Rodrigues, L., Miranda, H., Almeida, R., Jo a. M., and Vicente, P. (2002). The globdata fault-tolerant replicated distributed object database. In *Proc. of the First EurAsian Conference on Information and Communication Technology*, pages 426–433, London, UK. Springer-Verlag.
- [Schneider, 1993]** Schneider, F. B. (1993). Replication management using the state-machine approach. ACM Press/Addison-Wesley Publishing Co.
- [Son and Kouloumbis, 1993]** Son, S. H. and Kouloumbis, S. 1993. A token-based synchronization scheme for distributed real-time databases. *Inf. Syst.* 18, 6 (Sep. 1993), 375-389.