Simulation Modelling Practice and Theory 54 (2015) 64–85

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

SkyCDS: A resilient content delivery service based on diversified cloud storage

J.L. Gonzalez [a,*], Jesus Carretero Perez [b], Victor J. Sosa-Sosa [a], Luis M. Sanchez [b], Borja Bergua [b]

[a] Information Technology Laboratory, Center of Research and Advanced Studies of the National Polytechnic Institute (CINVESTAV), Ciudad Victoria, Mexico
[b] Carlos III University, Computer Science Department, Madrid, Spain

ARTICLE INFO

Article history:
Received 1 December 2014
Received in revised form 19 March 2015
Accepted 20 March 2015
Available online 6 April 2015

Keywords:
Content delivery
Multi-cloud storage
Pub/Sub overlay
Virtualization
Diversification
Risk management

ABSTRACT

Cloud-based storage is a popular outsourcing solution for organizations to deliver contents to end-users. However, there is a need for contingency plans to ensure service provision when the provider either suffers outages or is going out of business. This paper presents SkyCDS: a resilient content delivery service based on a publish/subscribe overlay over diversified cloud storage. SkyCDS splits the content delivery into metadata and content storage flow layers. The metadata flow layer is based on publish–subscribe patterns for insourcing the metadata control back to content owner. The storage layer is based on dispersal information over multiple cloud locations with which organizations outsource content storage in a controlled manner. In SkyCDS, the content dispersion is performed on the publisher side and the content retrieving process on the end-user side (the subscriber), which reduces the load on the organization side only to metadata management. SkyCDS also lowers the overhead of the content dispersion and retrieving processes by taking advantage of multi-core technology. A new allocation strategy based on cloud storage diversification and failure masking mechanisms minimize side effects of temporary, permanent cloud-based service outages and vendor lock-in. We developed a SkyCDS prototype that was evaluated by using synthetic workloads and a study case with real traces. Publish/subscribe queuing patterns were evaluated by using a simulation tool based on characterized metrics taken from experimental evaluation. The evaluation revealed the feasibility of SkyCDS in terms of performance, reliability and storage space profitability. It also shows a novel way to compare the storage/delivery options through risk assessment.

1. Introduction

Over the past ten years, the production of new data has dramatically grown [21]. Space agencies, hospitals, government instances, news agencies and scientific centers not only are producing huge amount of contents [20] but also they need to deliver them to different population segments through the Internet [15].

This type of organizations commonly builds content delivery systems (CDS) by using either web or FTP servers, which are a middleware between the organizations and end-users/customers [32,14]. Nevertheless, these servers are single points of failure as site outages produce a break in business continuity. Moreover, the servers can become bottlenecks because they also manage tasks such as distribution, coordination, and storage of contents. This increases the workload of the servers and produces delays on the end-user side when increasing the number of concurrent end-users.

* Corresponding author.

http://dx.doi.org/10.1016/j.simpat.2015.03.006
1569-190X/

Cloud storage approach is a cost-effective solution for organizations to build a dynamic and elastic CDS. Organizations can outsource either specific tasks or even the whole CDS with cloud providers to reduce the complexity of CDS management. Furthermore, the end-users also can retrieve contents anytime from anywhere by using almost any device [7,5].

Nevertheless, there is a lack of control over content management [16] when outsourcing CDS; as a result, organizations and end-users still have concerns about service outages [10,3], unauthorized mining of information [38] and lost content events [1]. Organizations and end-users are quite justified in expressing their concerns because they reject their legitimate expectation to privacy when voluntarily relegating private content to a *third-party* [36,2].

Vendor lock-in represents another obstacle for organizations to adopt cloud approach [38]. This problem arises when an organization contracts with a single cloud storage provider the management of all the contents, which produces a break in business continuity in two scenarios. In the first one, the organization decides to move the CDS to another cloud provider [4] and it is not clear that provider can manage the CDS in its current state because of software dependencies. In the second one, the cloud provider pulls out from the storage market [12] and the organization depends on the window defined by the provider for the clients to migrate their contents to another provider [11]. In both scenarios, the more content stored in the cloud the more migration costs [27,37,38,42]. Therefore, there is currently a need for mechanisms to ensure the service provision and to minimize risks of vendor lock-in situations when building a CDS.

In this paper, we present SkyCDS: a resilient content delivery service based on publish/subscribe overlay over diversified cloud storage.

SkyCDS splits the content delivery into two layers. The first layer is based on publish–subscribe patterns with which SkyCDS insourcing metadata control back to the content owner. This layer establishes the following roles: (i) Publishers, users that produce new contents, (ii) End-users, external customers that subscribe for contents, and (iii) Editors/administrators, organization users in charge of accepting/rejecting both publications and subscriptions. A resource management scheme enables the organizations to establish access control necessary for keeping up metadata flows as well as the flow of the Pub/Sub patterns between publishers and end-users.

The second layer is based on information dispersion [33] over a multi-cloud storage platform with which SkyCDS achieves efficient use of storage space and high reliability. In this layer, the dispersion is performed on publisher side by using resilient mechanisms called delivery workflows, which split contents into a set of redundant and anonymized chunks that are distributed on multiple storage providers. SkyCDS ensures that a given provider does not receive enough chunks to reconstruct original content. As a result, the way to rebuild the original content is kept on the publisher side.

The end-users are in charge of getting published contents by using retrieve workflows, which retrieve a subset of chunks and reconstruct contents on the end-user side. This means the retrieve workflows can get contents even when some cloud storage locations are unavailable. As a result, SkyCDS reduces risks of vendor lock-in and enables organization to outsource content storage in a controlled manner.

Fig. 1 shows the metadata and content flow layers of the SkyCDS overlay and how SkyCDS conducts the content delivery as a collaborative scheme. Fig. 1 also shows an example in which a publisher sends a publication of content $|C|$ to SkyCDS metadata flow layer (*Pub tag*). When an editor of the organization authorizes the publication of this content, the publisher is in charge of dispersing it to multiple cloud storage locations by using a delivery workflow (*Dy tag*), which runs on the publisher's computer. The content is added to the catalog and authorized end-users already can subscribe for the published and dispersed content (*Sub tag*). End-users with authorized subscriptions are in charge of retrieving contents by using retrieve workflows (*Retr tag*).

This overlay allows SkyCDS to minimize risks of vendor lock-in situations and scenarios in which there is a lack of control of critical procedures.

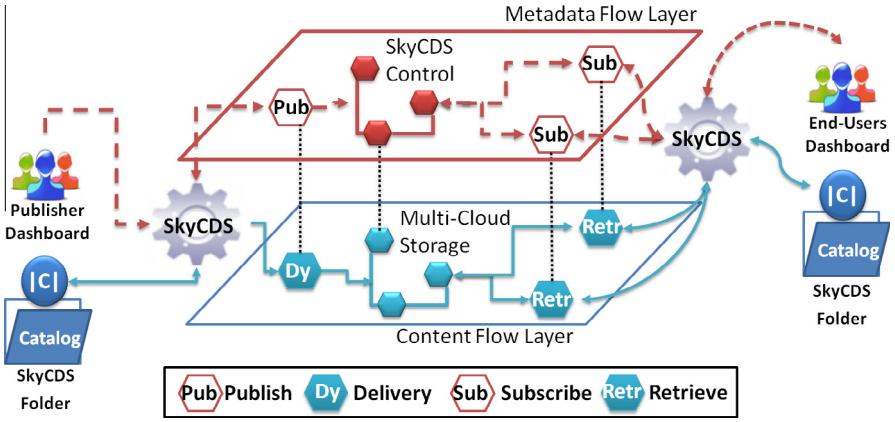

Fig. 1. The SkyCDS overlay.

The contributions of this paper are threefold:

1. *Collaborative content delivery based on a publish/subscribe overlay:* We developed a publish/subscribe RESTful model that enables end-users and publishers to collaborate in the content delivery process. It simplifies the management of resources and the control of the stages of the content delivery. The SkyCDS workflows are based on multi-core processing and continuous-flow transportation to mitigate the overhead produced on end-user and publisher side.
2. *A diversification strategy based on risk management for multi-cloud storage:* This strategy virtualizes the multiple storage locations available in an organization as a unified multi-cloud storage platform, which is simpler to manage than separate locations. A content placement method allows SkyCDS to perform allocation and location of contents in this platform to avoid putting all eggs/contents in the same cloud storage basket/location. This method distributes contents by using a risk assessment policy, which defines the risk level for each basket/location of the platform as well as a set of response actions for mitigating a reduced set of risks expressed by publishers and organizations about the content management in the CDS.
3. *Levels of Failure Masking:* SkyCDS includes three levels of failure masking. The first level masks service outages from end-users and publishers. This level is achieved by the dispersal information techniques applied to delivery and retrieve workflows. The second level masks organization site disaster from end-users. This level is based on a federation scheme in which a set of partners absorb the load of the organization during its outage. The last level masks the side effects of geo-diversity delays from all the users of SkyCDS by caching contents and moving them near to the end-users.

We deployed a SkyCDS prototype and compared the performance of delivery and retrieve workflows with different approaches such as a fault-tolerant distributed web storage service as well as a public and private content hosting service. The effectiveness of the diversification strategies was also validated.

In this evaluation we also conducted a case study based on data obtained from the European Space Astronomy Center (ESAC) for the Soil Moisture Ocean Salinity [9]. In this study, a set of organizations from two countries, spanning two continents, distributed satellite images through multi-cloud storage by using the SkyCDS overlay. The three levels of failure masking enabled a set of end-users associated to each organization to retrieve images in steady and faulty scenarios. Publish/subscribe queuing patterns were evaluated by using a simulation tool based on characterized metrics taken from experimental evaluation.

The experimental evaluation shows that the construction of a resilient content delivery service based on a publish/subscribe overlay over diversified cloud storage is feasible in terms of performance, reliability and profitability of the storage space. The evaluation also reveals that diversification and risk management strategies are a cost-effective solution for addressing a set of concerns expressed by publishers, end-users and organizations.

2. Related work

Content Delivery Networks (CDN) such as Akamai [17], Coral [19] or Globule [30] cache small pieces of information and distribute them to locations near who requests them; as a result, the end-users observe a reduction of latency and overhead in the content delivery process. Cloud providers take advantage of caching in content storage by using CDN services. Cloud-based storage services enable organizations to create catalogs of contents based on URLs and publishers/end-users can access them without simultaneous downloading restrictions by using any of web browsers, synchronizer based on HTTP streams or (S) FTP applications. However, organizations have no control of management of their contents when using this type of solutions [16] as the control is relegated to the cloud provider [36], which also relegates it to a CDN provider. In this context, this type of solutions puts organizations in high risk of not getting access to their contents during service outages [38].

In addition, the outsourced services are based on a pay-as-you-go pricing model that allows providers to establish rates based on the monthly stored contents plus the penalization stated at the service level agreement (SLA) contracted. These conditions could lead to long-term costs [27,37,42]. For instance, EUMETSAT and EOSDIS transfer around 1 TB of meteorological images per day, which might press organizations to consider an alternative service [37]. Moreover, in vendor lock-in scenarios, the migration of contents from a cloud to another is required and it could take from hours to a week for a volume of 12 TB [4] depending on providers.

Multi-cloud storage approach was proposed as a solution to solve the inconveniences of having one single endpoint for uploads, which associated to traditional cloud storage solutions. This approach offers multiple upload and download endpoints [40,39,29,26], so that uploads also take advantage from this benefit to deliver higher content availability.

The distribution of files on multi-cloud environments minimizes the side effects of storage service outages [13,40,39]. This kind of solutions are only available for public providers, hence solutions for private cloud models have also been proposed [29,26]. Solutions for mobile traffic [44,18] and distributed web storage have also emerged [25].

However, content delivery solutions that take into account both the user and organization concerns are not currently available. Risks assessment for distributing contents on a set of heterogeneous cloud services including private, federated and public models are also required by organizations.

Our Pub/Sub overlay represents an alternative to outsource storage content without outsourcing control. SkyCDS mitigates and even avoids the side effects of temporary or definitive cloud service outages by using a multi-cloud storage platform. It also withstands the arrive and departure of cloud storage locations by using both diversification and risk assessment

strategies. Although SkyCDS includes content caching mechanisms for reducing delays in content delivery, our solution differs from a traditional CDN in that SkyCDS was designed to deliver large contents to end-users instead small pieces of data/-files. SkyCDS ensures availability of contents such as satellite or manufactured images during large periods of time, which is required by organizations to ensure the storage and availability of contents produced by geographic information systems (GIS) and medical imaging applications.

3. SkyCDS: design and components

In this section, we describe the components and sub-components of SkyCDS as well as the roles of the users of the Pub/Sub overlay.

The catalog abstraction is the basic metadata component for organizations to establish controls of the content flows among users that produce new contents (publishers), external customers that subscribe for contents (end-users) and organization users in charge of accepting/rejecting both publications and subscriptions (Editors/administrators). In SkyCDS, the contents are allocated and located by using catalogs; as a result, only contents added to a catalog can be either published/delivered or subscribed/retrieved. When an organization creates a catalog, it also defines the attributes of a set of publishers that are allowed to add contents to the catalog as well as the groups of end-users that can subscribe for contents.

Client App is a component that enables publishers to add manufactured contents to the catalogs and disperse them to the multi-cloud storage platform. It also enables end-users to subscribe and retrieve published contents.

Pub/Sub and workflow engine agents are the components that are in charge of receiving and serving requests from Client Apps.

SkyCDS manager is the component in charge of controlling the Pub/Sub flows in the metadata layer.This component is also responsible for coordinating the content storage in the content flow layer.

Fig. 2 shows components of SkyCDS such as client app, agents, manager and engines. Fig. 2 also depicts the roles of publishers and end-users through an intuitive example that depicts the flows of metadata and contents in the overlay. In this example, a publisher adds the content $|C|$ to a catalog by using the Client App, which invokes a publish pattern (*Pub tag*). A Pub/Sub agent receives the request and performs access control tasks. The agent sends the authorized publication request to SkyCDS manager, which invokes an allocation task. In response, SkyCDS manager sends an authorization message to the engine agents chosen to serve the storage tasks and sends their locations to Client App with which it will disperse of content $|C|$. The client App invokes a delivery workflow engine that split content $|C|$ into n chunks ($C = \{c1 \ldots c5\}$ in this example) and transports them to the multi-cloud platform by using the information sent by SkyCDS (*Dy tag*). The engine agent sends a message of complete status to SkyCDS manager, which updates the list of the catalog publications. This list is also updated by the Client Apps of the authorized end-users, which already can invoke subscription patterns (*Sub tag*). The manager applies the same authorization process to retrieve workflows that applied to delivery ones. The authorized retrieve workflow (*Retr tag*) only retrieve k chunks suffice for constructing $|C|$ on the end-user side ($k = 3$ in this example).

It is important to note that the SkyCDS distributor agent does not send the whole content $|C|$ to a single cloud storage location. SkyCDS sends only codified chunks with anonymized names to different clouds, which manage them as regular files. The SkyCDS platform allows organizations to define the distribution strategy of the n chunks according to their organizational policies and available resources.

3.1. SkyCDS subcomponents: clients and agents

In SkyCDS, a Client App can take on different user roles (any of editors, publishers or end-users) depending on how the user is logged on in that App. This component includes a set of web dashboard applications for each type of user as well as

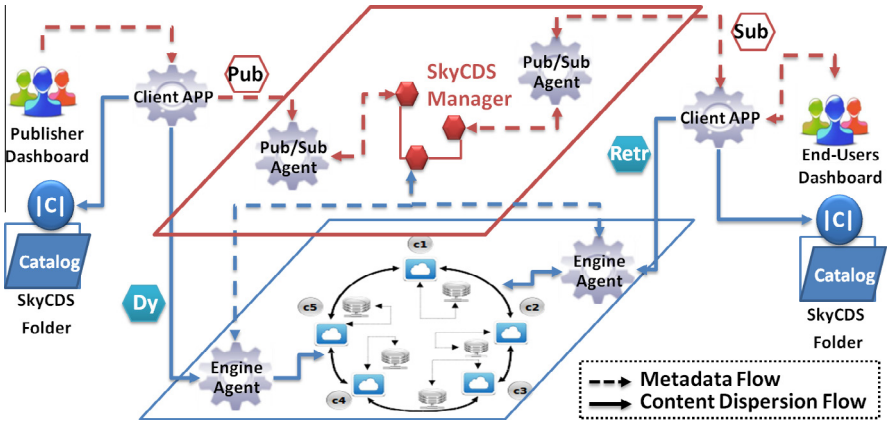

Fig. 2. The SkyCDS Components.

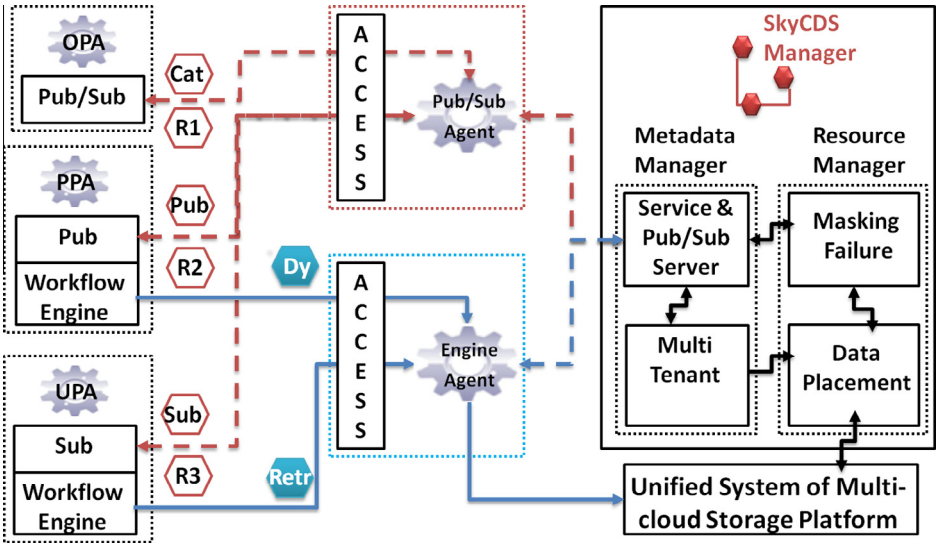

Fig. 3. The components and subcomponents of SkyCDS.

Pub/Sub and workflow engine tools. Fig. 3 shows all the subcomponents of SkyCDS and it will be used in this section to depict metadata flows among components and subcomponents.

- *Organization dashboard App (ODA):* This Client App enables editors to create catalogs, assign publishers to a given catalog and define subscriptions policies based on agreements and attributes. End-users from different communities or population segments can access a given catalog by using credentials with their attributes. An organization's administrator/editor also can define publication schedules per catalog. Fig. 3 shows an example of ODA creating a new catalog *Cat tag* and the set of responses sent by Pub/Sub engine indicating the status of publications and subscriptions *R1 tag*.
- *Publisher profile App (PPA):* This App enables publishers to obtain the list of catalogs in which they are authorized to add contents. PPA creates as many folders as catalogs in that list, hence publishers can add contents to the available catalogs by using either its profile in the dashboard or simply adding the contents to the folders created by PPA. This component also includes a daemon that is looking for new content in these folders, and when a new content is detected, the daemon invokes a Pub request (See *Pub tag* in Fig. 3). When PPA receives authorization from Pub/Sub agent (*R2 tag*), it invokes the workflow engine to build a delivery workflow (*Dy tag*). Although the organization defines the publication schedule, the engine disperses the contents to the repositories of the multi-cloud Storage Platform or (ULS) as a backup policy. As shown in Fig. 3, the engine agent sends a notification to the SkyCDS manager about when the dispersion operation is finished.
- *End-user profile App (UPA):* This App enables end-users to subscribe for contents from a list of published catalogs. When an end-user subscribes for a given content (*Sub tag*), this App invokes the workflow engine for building a retrieve workflow and to get the subscribed content(*Sub tag*). The engine agent notifies the state of this operation to the SkyCDS manager when such an operation is finished. This registration enables SkyCDS manager to perform tasks such as notification for publishers to create usage statistics and the creation of a trace route report of that content. An end-user can subscribe for a whole catalog and the UPA will automatically retrieve each new content added to the catalog whenever the operation is authorized (Content published, content subscribed). End-users can choose contents form catalogs by using a web dashboard. UPA also includes a list of contents already subscribed and available in the local host.

Each client/agent is provided by the SkyCDS manager with a valid *Authorization Token* for sending valid requests to the SkyCDS control Pub/Sub service. SkyCDS manager uses an *Authorization Token* as a control access and as a method to establish security policies.

3.2. SkyCDS manager

SkyCDS includes metadata and content managers. The metadata manager is in charge of Pub/Sub flows between the main components of SkyCDS and client/agents whereas the resource manager that is in charge of the allocation/location of contents and resource management.

3.2.1. Metadata manager

The manager includes the following modules:

- *Service subsystem.* This component is in charge of receiving request for *Authorization Tokens* as well as to receive publications and subscription orders coming from SkyCDS client/agents. The subsystem registers the transactions performed by publishers, organizations and end-users (See Fig. 3).
- *Publish and subscribe subsystem.* This module is in charge of the publication and subscriptions orders authorized by the service subsystem (See Fig. 3). A role attribute-based policy in this module enables SkyCDS to accept/reject publication and subscription orders. This module also includes an alert service that sends notifications to content publishers and organizations administrators about content subscriptions. It also sends notifications to end-user about new publications.
- *Multi-tenant subsystem.* This module manages the catalog and content properties as well as the user accounts by using a database. This module ensures that the contents of any publisher, end-user or administrator are isolated and remains invisible to other users. This module sends either allocation or location requests to the resource manager for a given content associated to a given catalog (See Fig. 3).

This manager is a critical component that is installed in-house in a cloud instance placed at a private cloud infrastructure. Fig. 3 shows an example of the metadata flows among agents, service, Pub/Sub and Multi-tenant subsystems. Fig. 3 also shows how the resource manager serves allocation/location requests sent by agents and authorizes workflow engines to transport data to a unified system (ULS). We will detail resource manager and ULS once workflow engine be described in next section.

3.3. Workflow engine

This component builds delivery workflows to transport contents from publishers' computers to the multi-cloud storage platform and retrieve workflows to transport contents from storage platform to end-users' computers.

The workflows are based on pipelines including two basic stages. The first phase is the encoding/decoding of contents, which is based on dispersal information algorithms (IDA) [33] and the second stage is the distribution of coded/encoded contents, which is based on continuous and parallel streaming [8].

3.3.1. Information dispersal algorithms

We encoding all contents by basically splitting a given content $|C|$ into n redundant chunks, which must be distributed to k different storage locations, so that every k pieces suffice for decoding $|C|$. As a result, it is granted that $|C|$ can be reconstructed when $n > k$ and the unavailable locations are $n - k$.

This algorithm can be implemented with different combinations of k and n parameters. This combination determines the codification costs in terms of storage space and computation time. The size of each resultant chunk is $|C|/k$, which results in a percentage excess of redundancy equal to $(n - k)/k$. Let us consider an IDA implementation with parameters $(n = 5; k = 3)$, in this case $|C|$ is transformed into five chunks and can be reconstructed by retrieving at least three chunks from any three different locations. The algorithm produces 66.7% of redundancy overhead when using this configuration, which is less than one replica.

Table 1 shows the amount of extra capacity spent for distributing n chunks to n different storage locations when requiring (at least) k chunks to support fault tolerance.

We performed an optimized version of this algorithm to save storage space in the content delivery process instead of using several replicas.

3.3.2. Multi-core processing and continuous-flow production

Although the dispersion algorithm presents a trade-off between the space consumption and the number of allowed failures, the encoding and decoding of the redundant chunks produces computation overhead whereas the chunk distribution produces latency. The overhead and latency in content delivery could affect the service experience of content publishers and end-users as the workflow engine is placed in their computers.

Table 1
IDA parameters combination k (chunks required for recovering files) and n (storage locations).

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
$n = 2$	100%					
$n = 3$	200%	50%				
$n = 4$	300%	100%	33.3%			
$n = 5$	400%	150%	**66.70%**	25%		
$n = 6$	500%	200%	100%	50%	20%	
$n = 7$	600%	250%	133.3%	75%	40%	16.7%

The SkyCDS workflows perform the encoding/decoding of contents by using parallelism and multi-core processing. This encoding/decoding technique takes advantage of all the processing power available in the computer where the engine is placed, which enhancing the performance of the encoding/decoding procedures. The production of each chunk is transported in a continuous flow to the multi-cloud storage platform (ULS) by using parallel streaming, which reduces the operations invoked by the workflow engine. This technique will be detailed in the description of each type of workflow.

3.3.3. Delivery workflow

The delivery workflow has been designed for publishers to disperse contents to n cloud storage locations. This workflow includes two metadata flows (publication and management) and one content flow (dispersion pipeline).

The workflow begins at the *publication phase*, when the PPA App sends both a request for delivering a content $|C|$ and for a valid *Authorization token*.

The workflow continues at the *management phase*, where the SkyCDS manager receives an *Authorization Token* from a publisher showing the interest of publishing a content $|C|$. The manager accepts/rejects the publication, and sends a request for storage locations to the resource manager subsystem and waits for a response. The manager produces a *service token* valid for ULS, determines n relative URLs mapping to n different cloud storage locations. The SkyCDS manager will response to the PPA with the *service token* and the n locations.

The final step happens in the *delivery phase*, where the PPA activates the workflow engine in dispersion mode. This engine builds a dispersion pipeline that considers three stages. In the *initial stage*, the engine introduces the storage locations (URLs) and the *service token* as input parameters, reads the content that will be dispersed, creates as many process as cores in the computer where the engine is running, and sends this configuration settings to the next stage, the *transformation stage*. In this stage, the engine splits the content $|C|$ into n chunks, encodes every chunk (adding redundancy) and sends them to the last stage, the *transport stage*. In this stage, n streams to relative URLs are created for the engine to transfer the different chunks. In the transport stage, the engine is also in charge of closing the active streams when the chunks codification and transmission ends and reporting errors that could arise. The PPA notifies the dispersion state to the SkyCDS manager that determines the publication schedule and sends alerts to end-users about this content.

Fig. 4 shows an example of the dispersion pipeline when the engine uses a $n = 5$ and $k = 3$ configuration.

3.3.4. Retrieve workflow

The retrieve workflow has been designed for end-users to retrieve contents from cloud storage locations by using a multi-threaded decoding procedure. This workflow includes two metadata flows (subscription and management processes) and a content flow (retrieval pipeline).

This workflow begins at *the subscription phase*, where the end-user through UPA App sends a subscription request for content $|C|$ to the SkyCDS manager.

In the (management phase), the SkyCDS manager verifies the *Authorization token*, authorizes/rejects the subscription for content $|C|$ and sends a request for k locations to the resource manager. The manager receives the request and determines k available cloud storage locations from the n used in the allocation of that content, and produces a *service token* for this operation valid for the ULS. SkyCDS manager responses to UPA with the *service token*, the k relative URLs that map to k different cloud storage locations, and the methods required to access them.

In the *retrieve phase*, a retrieve engine is activated in UPA, which is in charge of creating a retrieval pipeline that considers three stages. In the *initial stage*, the engine introduces the storage locations (URLs) and the *service token* as input parameters, writes a file with the name of the content C (that will be retrieved), creates as many process as cores in the computer where the engine is running, sends all the configuration settings to the *transformation stage* and waits for a response. In the

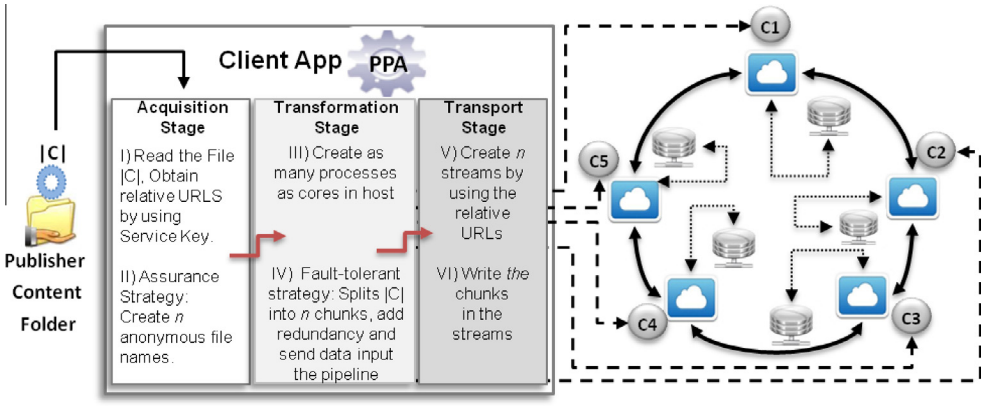

Fig. 4. Example of a dispersion pipeline with a configuration $n = 5, k = 3$.

transformation stage, the engine requests to the *transport stage* for k transmission streams associated to the relative URLs (storage locations) and waits for response. The *transport stage* creates the required streams, reads the corresponding chunks and returns the results to the *transformation stage*. In this stage, the engine reconstructs the content $|C|$ by using the k chunks, and sends the obtained results back to the *initial stage*, where each received bit is written in the file associated to the content C, using the local file system. Finally, when the decoding process of C is done the engine closes the active streams and reports to UPA the found errors, if any. UPA notifies the state of the retrieving process to the SkyCDS manager.

Fig. 5 shows an example of a retrieval pipeline with a configuration with $n = 5$ and $k = 3$.

Our implementation of the *transformation stage* is based on the Intel thread building block (TBB) technology [43,22,35] and the *transportation stage* was implemented by using the Curl libraries [8].

The multi-threading implementation of our dispersal/retrieve engines improves the performance of encoding and decoding tasks by taking advantage of multiple cores commonly found in current devices. This technique allows the engine to reduce the codification overhead making feasible to introduce a fault-tolerant scheme on the content publisher/end-user side. The implementation of the content delivery process as continuous flow allows the engine to avoid writing chunks in the local disks.

3.4. Resource manager: a diversification strategy for cloud storage

SkyCDS has been designed for managing metadata and the content storage in a separated manner. As a result, SkyCDS can outsource the chunk storage to different cloud storage locations considering different cloud models such as private, federated, public, or a combination of them (hybrid).

The SkyCDS resource manager is a middleware between the PPA/UPA Apps and the cloud storage services. The manager has control of the traditional I/O operations such as PUT, GET, LIST and DELETE enabled by the cloud storage providers. The manager preserves the LIST and DELETE functions and delegates PUT and GET to the PPA/UPA Apps. In order to avoid a chaotic dispersion of the contents increasing the storage management complexity, the manager includes three basic components. The first component is a unified cloud storage location system called ULS, which virtualizes the cloud storage locations either installed or acquired by a given organization, which is simpler to manage than separate locations. The second is a SkyCDS intermediary agent that receives requests from the publisher/end-user Apps in the form of Allocation/Location of contents. The last component is a data placement component (associated to ULS) that uses a method based on a diversification policy that converts allocation requests into n PUT operations and location requests into k GET operations. ULS is in charge of creating maps to valid and available cloud storage locations that can receive operation defined by the data placement method.

3.4.1. Unified storage of multi-cloud platform (ULS)

The ULS subsystem performs four tasks: the first is to centralize the control of multiple cloud storage locations. The second is to assign *service tokens* to I/O operations defined in a diversification strategy, and the third is to produce maps to valid cloud storage locations that are used by SkyCDS Client Apps to disperse and reconstruct contents.

The ULS includes a monitor that verifies the availability of each storage path as well as the consumption of the storage resources associated to each path.

The ULS subsystem registers in a database the access methods of each storage path, the metrics for path management such as the storage quotas for paths from a private model, agreement characteristics for federated paths and SLAs and type of services (ToS) of public paths. This database also stores the information captured by the monitor for keeping the ULS statistics updated. The ULS subsystem also registers the operations performed by SkyCDS with each storage path.

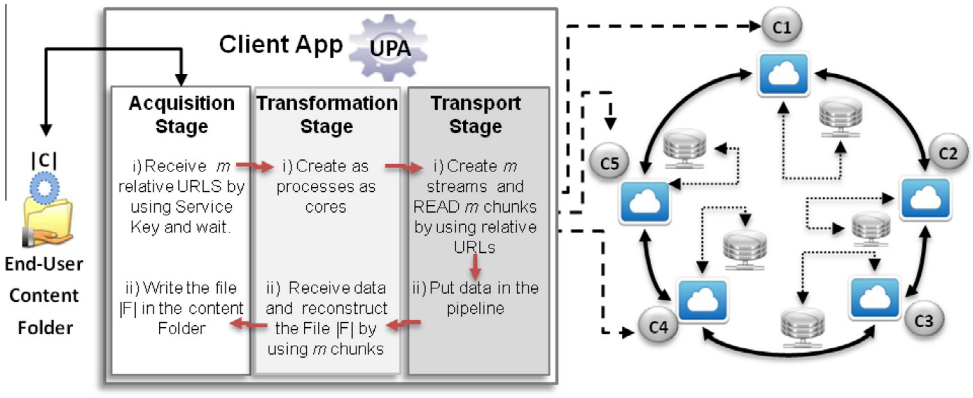

Fig. 5. An example of a retrieval pipeline with a configuration $n = 5, k = 3$.

3.4.2. DPM: a data placement method based on diversification strategies

The data placement method (DPM) converts one content allocation request into n number of PUT operations and one location request into k GET operations. The main goal of DPM is to choose either n or k locations from multiple locations based on a diversification policy.

Diversification is a resource allocation technique designed for enhancing the resources profitability and to ensure the business continuity. This technique is based on the definition of contingent and mitigation plans (road maps) that describe how to distribute critical components for ensuring that either they will operate even during incidents and disasters or they will be recovered to a previous operational state in a reasonably period of time. This technique has been a proven risk-reduction solution in the financial sector [34].

3.4.3. Management and assessment of risks

Our diversification policy is based on an abstraction called portfolio. This abstraction organizes the storage resources in stocks such as private, shared and public. The stocks include a set of available storage paths that DPM can choose for serving allocation and location requests.

This policy assigns a set of risk labels to each stock based on a set of inherent risk calculated for each stock. DPM takes the risk labels into account when choosing paths for serving allocation requests.

In this policy, the risk labels are separated into operational and safety risks.

The operational risks may impact on the content delivery service and they can be labeled with either (P) when the performance is affected by computational overhead and/or latencies. This is a risk affecting content publishers and end-users as they are assuming the content flows. When a stock produces delays in the managing of metadata, a performance degradation could be observed on the organization side s they. In this case, DPM can use the label (M) for identifying this type of risk.

The security risks of a storage path may impact on the integrity of the contents. The (LG) label represents the risk that arises when a content provider stores a given content (not portions) by using locations geographically dispersed. This could result in a legal issue because the content management is under the control of a third party. The (Pi) label represents a privacy risk when either the organization shares the content management or when it simply has no control of the storage path where the contents are stored. This is a risk that basically affects to content providers and organizations. The label (A) is used when content availability is affected by failures of a given storage path. This is a risk for end-users, publishers and organizations.

Each label in each stock has a different risk level, which should be organized in either a list or a matrix structure. A matrix of risk and frequency [41,28] is the most used in financial environments. In this type of matrix, the rows are risks, very low, low, medium, high and very high, while the columns are frequency such as few probable, probable, very probable and highly probable. This type of matrix could also express a given consequence or impact instead of frequency.

In the matrix generated by our policy, the risk levels are expressed as rows such as low, medium/regular and high, while columns represents the stocks. The value in the interception between a row and a column is the corresponding risk label. A low risk indicates that a stock offers the lowest risk for a given label and it is used for sensitive information. A regular risk indicates that some inherent risk could be assumed and a high risk could be used for non-sensitive contents.

3.4.4. Heuristics for risk management

DPM creates a ranked list of the risk labels per stock. By now, this ranking is static and has been performed by experimentation that will be detailed at the results section.

This ranking allows organizations to perform a risk assessment with which they can define road maps to mitigate as side effects as possible. In the very early stages of the content delivery process, the publishers can express their concerns about the management of their contents by adding labels to the content before sending them to the SkyCDS overlay. For instance,

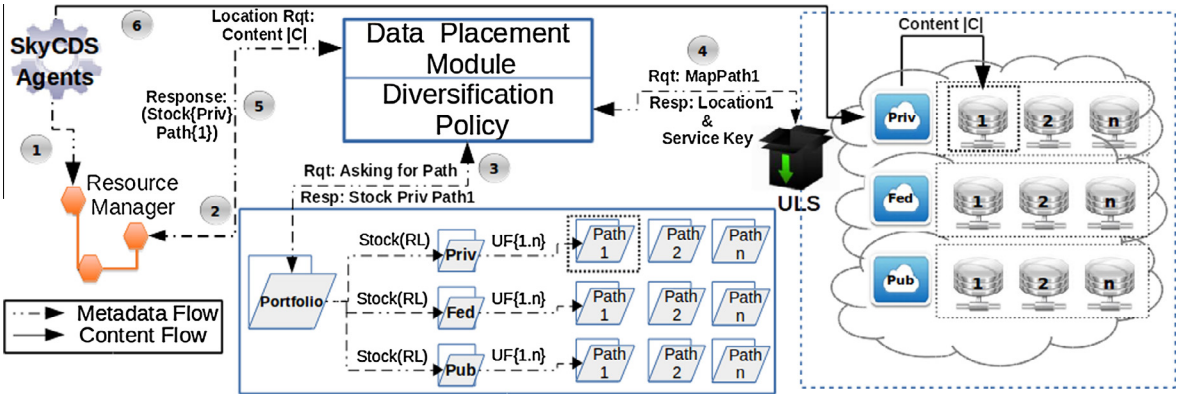

Fig. 6. An example of data allocation in the SkyCDS resource manger.

when a content owner sends a risk label (P), DPM should choose a stock (cloud storage location) mapped with a low risk label (P), which will offer the best response times as the priority of the owner is the performance.

Fig. 6 shows an example of an allocation/location operation in which an end-user sends a location request to the ULS for storing content $|C|$ (1). The content was labeled with (P) and DPM must choose the stock that offers a low risk level (2). In this example, the stock representing a private cloud has the lowest risk for label (P) in the risk ranking of the portfolio. When DPM has chosen the stock (3), ULS determines which k paths are available by using a distribution policy (4). In this example, the DPM has been configured for distributing contents by choosing the path with the minimum value obtained from a usage function (UF), which is a number between 0 and 1 that represents the used portion of a given component [26]. The UF can be calculated by using weights produced either by round robin or pseudo-random policies as a load balancing method. The SkyCDS resource manager responses to the SkyCDS end-user App (5) with the information required to perform a retrieve workflow (6). For simplicity reasons, this example only shows the location of one chunk.

In order to define the risk ranking, we consider one heuristic for each risk label with which DPM calculates a risk level per stock. For instance, the ranking of label (P) is defined by using the registers on the database of the ULS monitor to calculate the mean response time per path $PathR(P)$ and per stock $StockR(P)$.

$$PathR(P) = PS + AL + Stream$$

where PS represents the time spent in Publish–subscribe operations, while AL the time to make decisions about content location, $Stream$ is the time required to store the content.

$$StockR(P) = \sum_{i=0}^{n} PathR/n.$$
$$top = Max(StockR - i(P)..StockR - n(P))$$
$$bottom = Min(StockR - i(P)..StockR - n(P))$$
$$StockLevel(P) = 1 - (top - bottom/top)$$

where top and $bottom$ represent the worst and best response time respectively. $StockLevel(P)$ represents a value between 0 and 1, which is used for making a simple level ranking for a given stock, for the label (P) in this case. As a result, a stock is

$$high = StockL(P) > \frac{2}{3};$$
$$regular = StockL(P) > \frac{1}{3} \ and \ < \frac{2}{3};$$
$$low = StockL(P) < \frac{1}{3}$$

In the case of label (M),$PathR(M) = PS + AL$ as $Stream$ is not a metadata operation. In the case label (A), $Stock(A)$ represents the number of unavailable site/domain per stock without service interruption. In this case, the number of allowed failures determines the $StockL(A)$. DPM defines $Stock(Pi)$ and $Stock(LG)$ according to the control degree that an organization has in each stock. The less control the high risk level.

This risks levels for each label represents the heuristics required by the data placement to define the more suitable stock to receive a given content/chunk. The risk matrix will be described in the experimental section.

4. Levels of failures masking

SkyCDS includes three levels for masking failures of CDS components, which can be chosen by organizations to configure their CDS.

The first level masks service outages from end-users and publishers. This level is achieved by the dispersal information techniques applied to delivery and retrieve workflows.

The second level masks organization site disaster from end-users. This level is based on a federation scheme in which a set of partners absorb the content load produced by publishers and end-users during outages of the site of their organization.

The last level masks the side effects of geo-diversity delays from all the users of SkyCDS. This level of failure masking is based on caching of contents and moving them near to the end-users.

In SkyCDS, organizations can build federations based on partnerships. In this federation, each partner defines the amount of publications and subscriptions orders that could be served when a member is unavailable. Each Pub/Sub server has a partner in the federation and the SkyCDS UPA and PPA Apps are configured to send publications and/or subscriptions to the SkyCDS manager of their organization as the primary option, while a partner represents a secondary option. The monitor in the resource manager enables the SkyCDS Pub/Sub agents to receive publications/subscriptions orders from UPA and PPA Apps of a partner. The monitor sends a notification of the clients App to inform that its new SkyCDS manager will receive requests for a negotiated period of time. When the unavailable SkyCDS manager comes back to the service, it informs to its partner and client Apps. In this case, the failure transparency depends on the negotiation model.

The third way of failure masking arises when an organization requires a given content that has been published by a partner for delivering it to their end-users. In this case, the organization sends to its partner a retrieve workflow to get that content. In the meanwhile, the organization retains the publication of the content. When its cloud storage infrastructure has all the corresponding chunks, the organization performs the publication and the end-user will observe a regular response times as the organization is absorbing the bandwidth costs. This procedure moves the chunks near to the client Apps to improve the service experience of the end user. In this case, the failure transparency depends on the publication schedule defined by the organization.

Organizations can adapt their CDSs to their available cloud resources by choosing the levels of failure masking to be used in the CDS. For instance, when this organization only has available storage locations and compute instances in private and public clouds, an organization can configure its CDS with failure masking levels one and three. Moreover, an organization could configure its CDS with the failure masking level two when the organizational structure includes partnerships and a set of partners agree to cooperate to advance mutual interests such as the withstanding of organization site outages or site disaster of a set of partners.

5. The prototype

We considered a scenario where a set of organizations build a CDS based on our Pub/Sub overlay. The three organization are represented by the following acronyms *UC3M-Colme*, *UC3M-Cloud* and *Cinvestav*. *UC3M-Cloud* is located in Leganes, *UC3M-Colme* is located in Colmenarejo (both cities near to Madrid, Spain). *Cinvestav* is located in Northeastern Mexico. These organizations configured their CDSs with three failure masking levels.

Each partner is running a SkyCDS manager image and are connected forming a cyclic list. We have launched a set of SkyCDS client images in the infrastructure of all the members with which the end-users retrieve the contents published by one publisher per organization.

All the components of SkyCDS of *UC3M-Cloud* and *Cinvestav* organization were based on cloud instances built with OpenStack. The SkyCDS component of *UC3M-Colme* organization were built by using physical computers for performing a set of tests from the perspective of end-users and publishers.

Table 2 shows the features of the storage infrastructure of each organization as well as the clients included in this prototype.

6. Experimental and simulation scenarios

We defined two evaluation scenarios for testing performance, metadata management overhead, and profitability of storage resource of the SkyCDS. The first scenario has been defined for evaluating the side effects of SkyCDS on the publisher, end-user and organization sides in each phase of the content delivery service (from encoding/decoding to data placement decisions).

The second scenario has been conducted as a study case based on information about satellite images of an space agency in which was also evaluated the different ways of failure masking in a whole content delivery process through a federated implementation of SkyCDS.

Pub/Sub queuing patterns were evaluated by using a simulation tool based on characterized metrics taken from experimental scenarios.

6.1. Workload emulation

We developed an *IO_Launcher* process for producing publishing, subscription, delivery and retrieve workflows operations. The *IO_Launcher* was included in the cloud image of the SkyCDS client Apps. The *IO_Launcher* was installed in the same machines where are the client Apps in order to send requests to the SkyCDS manager, which assumes this artificial load comes from real and valid users. The *IO_Launcher* captures the metrics for each operation performed in each component of the SkyCDS (agents, clients, metadata manager, ULS and resource manager).

Table 2
Characteristics of client agents.

	PCs, cloud instances	Cores	RAM
Agents			
UC3M-Cloud	5 Instances	2	4 GB
UC3M-Colme	2 PCs	4 (i7) and 2 (i5)	4 GB
Cinvestav	5 Instances	4	8 GB
Clients			
UC3M-Cloud	2 Instances	4	4 GB
UC3M-Colme	2 PCs	7 (i7) and 4 (i5)	4 GB
Cinvestav	5 Instances	2(3) and 3(2)	2(1 GB) and 3(4 GB)

As comparative scenarios, we also implemented *IO_Launcher* to send upload and download operations to a fault-tolerant distributed web storage, AmazonS3 [6], a File Hosting System (FHS) and a distributed cloud storage (DCC).

The *IO_Launcher* in the client machines can be configurable to produce workloads with different distributions and features such as inter-arrival, content size and type of operation (Pub/Sub and Deliver/Retrieval).

6.2. Metrics

The *IO_Launcher* and metadata manager captured the following metrics:

- *Encoding/Decoding time:* This metric helps us to determine the costs of the redundancy production.
- *Service time:* This metric helps us to determine the costs assumed by the organization. This time is measured from the Pub/Sub request arrival to the metadata manager until the time point in which the SkyCDS manager sends its response to the client App.
- *Response time:* This metric helps us to determine the degree of satisfaction for end-users and publishers. This time is measured from the Pub/Sub moment of a given content until the time point in which a SkyCDS client retrieves that content, meaning that request has been successfully dispatched. This time includes the service time and the streaming time, which also includes the network round trip latency and the write/read time in the temporal paths as well as the encoding/decoding time.
- *Overhead of Failure Masking:* This metric helps us to determine the costs to keep failure transparency.

6.3. Pub/Sub queuing simulation

In order to reduce time and resources in the estimation of the overload scenarios on the components of SkyCDS, we connected our workload launcher with a Pub/Sub queuing simulator inspired in the very principles of task scheduling reported by Gkoutioudi [23]. The launcher builds traces that consider the mean inter-arrival time, the mean number of Pub/Sub patterns and the number of simultaneous requests accepted by simulated SkyCDS component. It also builds a configuration file that indicates the component to be stressed and the diversification/distribution policies with which the workflows will be invoked. The launcher sends the trace and configuration files to a Pub/Sub queuing simulator instead to launch delivery/retrieve workflows through the multi-cloud storage platform. When the simulator receives both files, it calculates the mean delay for Pub/Sub patterns of the trace by using the mean service time registered in the SkyCDS database for each scenario indicated in the configuration file. This type of simulation allows organizations to estimate, in real time, the production rates on the publisher side, the consume rates on the end-user side, and the overload on the Pub/Sub metadata servers on the organization side.

The SkyCDS prototype can work as a benchmark for obtaining characterized service times from different delivery content scenarios and as a Pub/Sub queuing simulator to estimate saturation, overload and overhead during high rates of either production or consumption of contents.

7. Experiments and results

As we already said, we defined two scenarios, the first to evaluate each component of the SkyCDS and the second for evaluating the whole service through a study case.

7.1. Performance analysis of the SkyCDS components

In this section, we evaluated encoding/decoding, response and service time metrics in each phase of the content delivery service.

7.1.1. Performance analysis of encoding/decoding processes: the client App perspective

This section presents the evaluation of the parallelism efficiency of the multi-threaded workflow engine. We used a pair of SkyCDS client Apps installed in physical computers of the *UC3M-Colme* organization and three configurations were defined: (i) *Serial-nocache-I7*, this is an encoding process from a serial version of the information dispersal algorithm (IDA). (ii) *tbb-nocache-i7*, this is the encoding process performed by our multi-threaded workflow engine running in UMA architecture. The *tbb-cache-i7* configuration is an encoding version that enables the hot mode when using the cache. (iii)

Table 3
CPU architecture details.

CPU specification	Sockets	Cores per socket	Hyperthreading	Memory (GB)
Intel(R) Core(TM) i7-2600 CPU @ 3.40 GHz	1	4	Yes	8
Intel(R) Xeon(R) CPU E7-4807 @ 1.87 GHz	4	6	Yes	128

tbb-nocache-c48, this is our encoding multi-threaded version for NUMA architectures. An encoding configuration *tbb-cache-c48* with enabled cache was also evaluated.

Table 3 shows the CPU features of the computer architectures UMA (i7) and NUMA (Xeon) included in the evaluation of the tested configurations.

We performed a set of experiments where the *IO_Launcher* sent requests to the client Apps to build delivery and retrieve workflows by duplicating each time the content size from 1 MB to 1 GB. The *IO_Launcher* also captures the encoding/decoding time produced by the workflows.

Fig. 7 shows, in the vertical axis, the chunk encoding time and dispersion of chunks to local hard disks for all the tested configurations. The size of the contents is shown in the horizontal axis. Fig. 7 shows that the chunk encoding time produced by the configurations multi-threaded workflow engine are significantly better than the sequential version of the algorithm (serial). When the configuration is in hot mode (tbb-cache), the results are improved according to the memory as well as the number of cores of the computer. NUMA architecture improves the results of the UMA architecture as it takes advantage of the increased number of cores (NUMA platform disposes a number of cores x8 than UMA platform). Nevertheless, this is not determinant for obtaining a significant improvement in comparison with the serial version; as a result, a regular computer with UMA architecture on the publisher and end-user could mitigate the encoding costs.

The streaming of contents are not included in the metrics as we evaluated the parallel efficiency in this set of experiments.

Fig. 8 shows that decoding produces a similar behavior to the encoding procedure. Nevertheless, in the decoding procedure the memory is, at a given point, a factor to improve the performance.

The experiments shows the benefits of taking advantage of the computing resources of end-users and publishers. Nevertheless, the speed up and parallel efficiency of our codification technique must be also evaluated.

We calculated the parallel efficiency of our technique by determining the speed up ratio of elapsed time for 1 and n workers (cores).

$$SpeedUp = t_1/t_n$$

The parallel efficiency ($E = SpeedUp/n$) for all the configurations is shown in Fig. 9. Almost all the multi-threaded configurations show a $E = 1$, which results in our multi-threaded version scales linearly.

7.2. Delivery and retrieve workflow performance evaluation: the publisher and end-user perspective

In this section, we evaluated the SkyCDS performance from the publishers and end-users perspective by testing delivery and retrieve workflows.

We compared the SkyCDS workflow performance with a web fault-tolerant distributed storage system and a solution based on a public cloud storage publisher and a distributed file Hosting Service.

We designed a synthetic workload in which *IO_Launcher* sends an incremental load by duplicating the content size from 512 KB to 1 GB. We captured, for each request, the response time (metadata management + encoding/decoding + content streaming + storage).

We defined the following configurations:

- *Private(DistributedFHS):* In this configuration, users store and retrieve contents by using a Web File Hosting System (FHS). This is the traditional solution to deliver contents to end-users. We have implemented a *Private FHS* in *UC3M-Colme* and *UC3M-Cloud* organizations in which the contents are stored without producing and distributing redundancy.
- *Public(AmazonS3) [6]:* In this configuration, users store and retrieve contents by using an AWS Amazon instance with the standard redundancy associated to a free account.

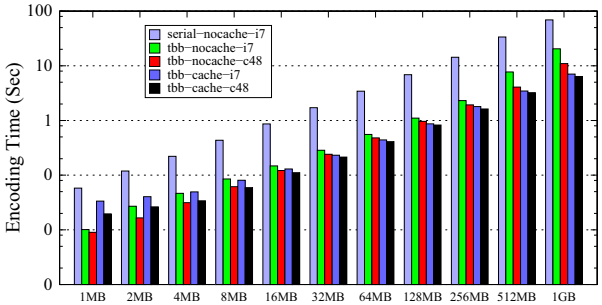

Fig. 7. The encoding time with a configuration ($n = 5, k = 3$).

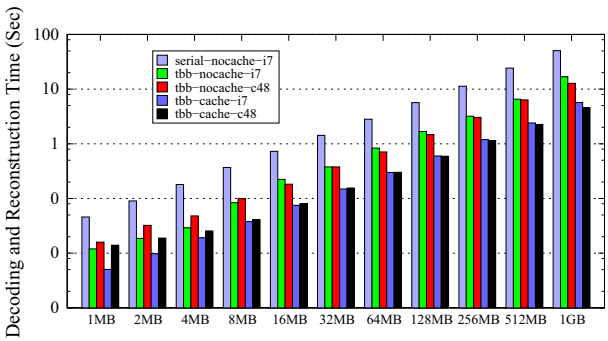

Fig. 8. The decoding time with a configuration ($n = 5, k = 3$).

- *Fed(DistributedDispersion):* In this configuration, users store and retrieve contents by using an Online Distributed Web Storage System called Phoenix [25]. We implemented this system in the *UC3M-Colme* and *UC3M-Cloud* organizations and it has been configured to apply a fault-tolerant strategy to the contents by using dispersion through the serial IDA algorithm. As a result, this configuration allows the CDS to withstand two metadata agents failures and one site failure.
- *SkyCDS:* We implemented our SkyCDS in the *UC3M-Colme* and *UC3M-Cloud* organizations. This configuration allows the CDS to withstand two metadata agents failures and one site failure and allows us to measure the performance in all the stages of the delivery and retrieve stages.

Fig. 10 shows, in vertical axis, the mean response time obtained when a publisher sends contents of different sizes (horizontal axis) to a cloud storage location.

Fig. 10 also shows, in log 10 basis, that latency and delays are the more important factors for the service experience of the publishers. For instance, *Private(DistributedFHS)* yields the best performance because it returns the control to the user when the content arrives to the cloud without applying any fault-tolerant strategy to the content delivery. As a result, *Private(DistributedFHS)* does not generate delays for redundancy overhead, which improves the response time observed by the publishers. Moreover, *Private(DistributedFHS)* is situated in Colmenarejo and Leganes and the clients were distributed on both cities. This scheme reduces latency because both cities are geographically close to each other. Nevertheless, one single failure in this type of system results in lost data.

Public(AmazonS3) configuration performs the same procedure as *Private cloud(FHS)* but it produces latency overhead because its servers are located at Ireland and the clients are sending requests from Spain. Moreover, the end-user cannot access the content when the service or the domain is unavailable.

Fed(DistributedDispersion) configuration produces the worst delay because it solves the vulnerability window in which the user could loss data by immediately splitting contents into chunks and distributing them to cloud storage locations of the two organizations. Once this has been performed, *Distributed (Federated storage)* returns the control to publishers.

The *SkyCDS* performance is better than *Public(AmazonS3)* configuration (44% in mean) when the content size <4 MB because the locality compensates the overhead of codification and, when the content size >4 MB, *Public(AmazonS3)* configuration is better than *SkyCDS* (9.14% in average) because *SkyCDS* distributes five chunks per I/O request (66.7% more data than *Public(AmazonS3)*). Nevertheless, this increment represents a good deal for publishers because *SkyCDS* ensures the content availability in a faulty scenario and for organizations as this configuration optimizes the storage space for redundancy overhead 66.7% against 100% when using a single replica.

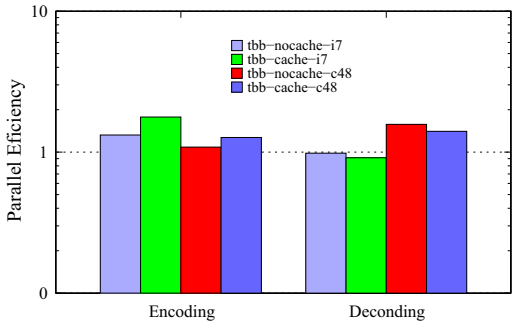

Fig. 9. Parallel efficiency of multi-threaded configurations.

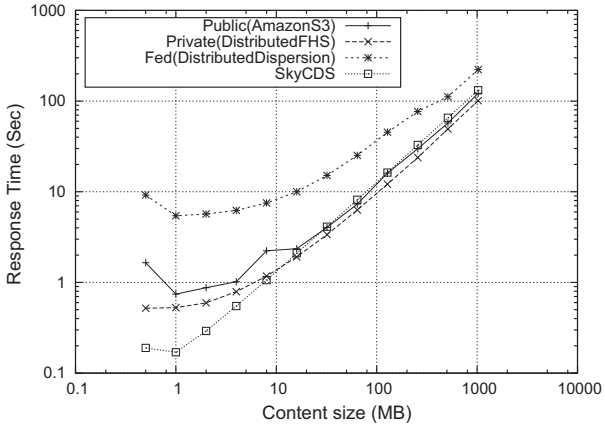

Fig. 10. The response time on the publisher side.

Fig. 11 shows, in log 10 basis, the results obtained when end-users retrieving contents by using a *SkyCDS* client App. We observed that all configurations take advantage from different streaming policies of ISP (downloads have more bandwidth than uploads).

In addition, *SkyCDS* also improves its performance as the client App only retrieves $k = 3$ chunks in the retrieve workflows while delivery workflows distributes $n = 5$ chunks. The performance of *SkyCDS* is quite similar to the *Public(AmazonS3)* and even better in average than this configuration. Moreover, the failure of storage locations was transparent for end-users, which only is provided by *Fed(DistributedDispersion)* but at a higher overhead in the response time.

Considering the security and performance risks, we can see that the risk level of a performance overhead (P) in *Private(DistributedFHS)* is low, regular for *SkyCDS* as it also reduces latency by using a portion of the private infrastructure and high for *Public(AmazonS3)* when its data center is far from the publishers.

SkyCDS offers the same reliability as *Fed(DistributedDispersion)* at reasonable cost. Moreover, *SkyCDS* preserves the original content in-house, which means the content cannot be reconstructed by the administrators of a given organization without obtaining $(k - 1)$ chunks from either other organization or the publisher.

In this case, we observed that the risk level for content unavailability (A) in *Private(DistributedFHS)* is high, regular for *Public(AmazonS3)* and low for *SkyCDS*.

7.3. Performance evaluation from the organization perspective

In this section, we evaluated the performance on the organization side by measuring the service times.

Fig. 12 shows, in log 10 basis, the service time produced by delivery workflows for different content size when using the studied configurations.

Fig. 12 also shows that the difference between service time of *Private(DistributedFHS)* and *SkyCDS* is not significant. *Private(DistributedFHS)* configuration only spends time in register of the content operation in the database of the metadata

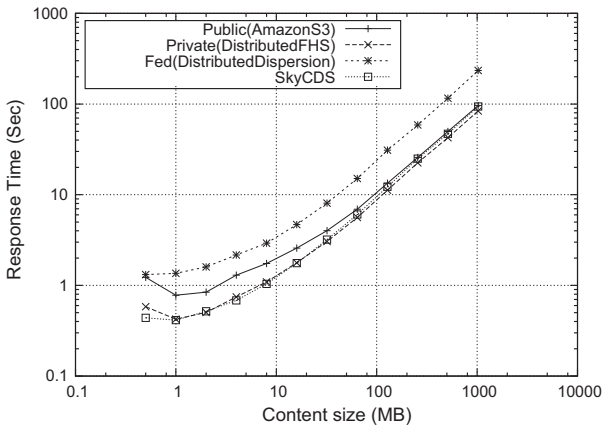

Fig. 11. The response time on the end-users side.

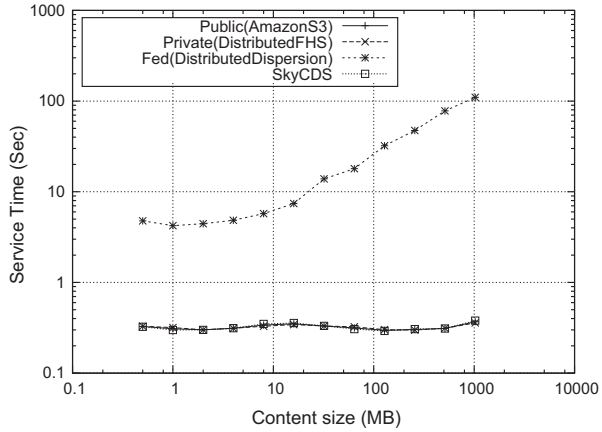

Fig. 12. The service time for delivery workflows.

manger while *SkyCDS* spends time only in the synchronization of agents, client Apps, resource manager and the registration of *n* chunks (332 ms in average).

Public(AmazonS3) only adds overhead when sending the token and credentials, which it is quite similar to *SkyCDS* mainly because of latency issues.

Fig. 12 also shows the overhead when *Fed(DistributedDispersion)* configuration avoids a vulnerability window by dispersing contents on the fly. *Private(DistributedFHS)* configuration eventually also will introduce an overhead when creating and distributing redundancy of contents in deferred manner. The *SkyCDS* configuration reduces this overhead as this work is performed by the client App on the publisher side.

Fig. 13 shows the service time produced by all the configurations when the end-user retrieves contents of different size.

Private(DistributedFHS), *Public(AmazonS3)* and *SkyCDS* configurations produce the same behavior as observed in Fig. 13. In this case, *SkyCDS* and *Private(DistributedFHS)* only spend time in queries to the metadata manager for determining storage locations, which represents 132 ms in average. In the case of *SkyCDS*, this load is distributed among the agents of the metadata manager. It is worthy to note that the time spent to write/read in disk is included in the content streaming for these configurations as this time has been included in the response time and it is not considered in the service time. In the case of *Fed(DistributedDispersion)* configuration, the decoding and reconstruction of the contents are performed on the organization side, which produces the overhead shown in Fig. 13.

Taking into account the delivery and retrieve processes, we observed that the risk level for content management overhead (M) in *Private(DistributedFHS)* is high as it eventually will perform redundancy tasks, regular for *SkyCDS* as the more users the more traffic, and low for *Public(AmazonS3)* as the metadata costs are reduced when sending tokens and credentials.

Fed(DistributedDispersion) is not taken into account as *SkyCDS* is an optimized version of that federated fault-tolerant storage system.

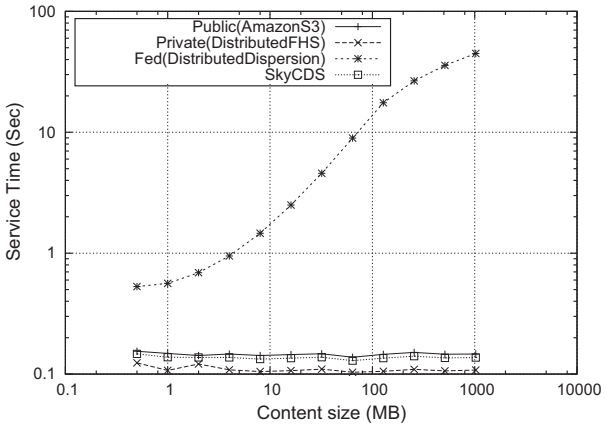

Fig. 13. The service time for retrieve workflows.

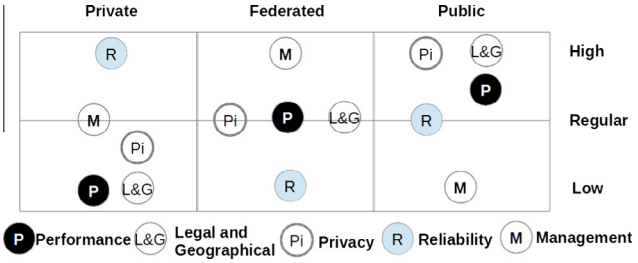

Fig. 14. The risk assessment matrix based on labels per stock.

7.3.1. Performance evaluation of risk labels in Workflows

We assigned the heuristics based on experimentation shown in previous section for building the risk matrix shown in Fig. 14.

Fig. 14 shows the matrix of risk ranking, calculated by the DPM when finished the performance tunning. This matrix includes heuristics for each stock (private, federated, and public) and each label: Performance (P), Management (M), Availability (A), Privacy (Pi) and Legal and Geographical (LG). In private stock, the labels (P) and (LG) where ranked in low risk as in this stock the organization has a best control level in comparison with federated and public models, (Pi) in regular and (A) and (M) in high. In federated stock the label (A) was ranked in low risk, (Pi, P, LG) in regular and label (M) in high. In the Public stock the label (M) was ranked in low risk, (R) and (P) in regular and (Pi, LG) in high.

The distribution of contents by using any model could result in a set of risks that the organization should assume. A combination of labels could be used for distributing load on private, federated and public stocks by using the SkyCDS workflows.

In this section, we introduced labels into the DPM of the SkyCDS for measuring the (P), (M), (A) labels in low risk level.

In this experiment, SkyCDS sends all content traffic to the private cloud (*Private100%*), when the catalog is published with a label (P), with (M) to the public (*Public100%*) and with (A) to the Federated (*Fed100%*). When (P) and (A) labels are randomly assigned to the contents of a catalog, the traffic is sent to Private and Federated (*Priv50%Fed50%*) stocks, when using (A) and (M) to (*Fed50%Pub50%*) and when using (P) and (M) to (*Priv50%Pub50%*). When (P)(A)(M) are chosen, the traffic goes to (*Priv33%Fed33%Pub33%*).

SkyCDS provides a masking of cloud storage location as all contents are delivered by using SkyCDS encoding workflows. When the client APP adds the label (A), it means the SkyCDS also has been configured with failure masking level two for the contents allocated in the Federated stock.

We sent synthetic content traffic for these configurations and measured the response times, which are shown in Fig. 15. As expected, the configurations take the risks of the stocks and combing them. For instance, the best performance (P) is delivered by (*Private100%*) but this catalog has a high risk for label (A). The best performance and management (P), (M) is delivered by (*Priv50%Pub50%*), while the best trade-off is achieved by *Priv33%Fed33%Pub33%*. As a result, organizations can configure road maps for publishers to enhance the profitability of storage resources available in the organizations. In this context, the publishers and organizations could define when a low risk for a given label can be assigned to a given catalog.

The behavior of the delivery and retrieval of contents is quite similar as shown in Fig. 16.

We defined a road map matrix from literature in our experimentation, which could be applied to a diversified multi-cloud storage, which is shown in Table 4.

We also simulated the Pub/Sub patterns queuing of the tested policies by using the simulation tool described in Section 6.3. The characterized metrics that were shown in Fig. 16 were used by this simulator to perform a set of experiments.

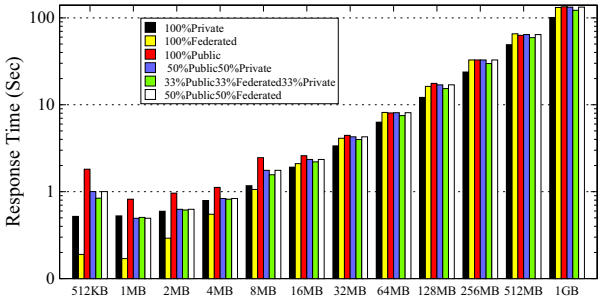

Fig. 15. The response time of diversification policies per file size.

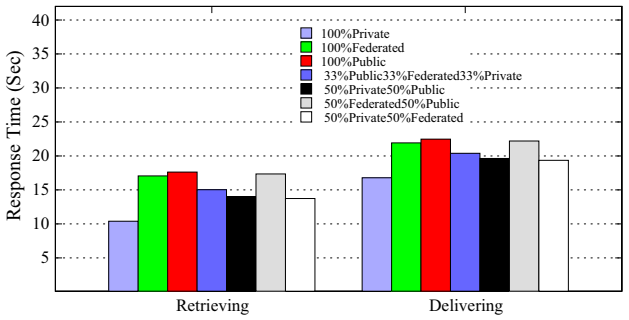

Fig. 16. The average response time of the delivering and retrieving of contents per diversification policy.

Table 4
Risk analysis and road maps.

Concern about	Metrics	Risks	Mitigation	Side effects
Performance	Latency, service and response time	Overhead [16]	Distribution, parallelism and continuous flows [29]	Complexity and costs
Management	Service time	Overhead and security [16]	Distribution, aggregation, encryption and federated identity management	Complexity, costs and negotiation schemes
Availability	Number of failures supported per stock	Blackouts, and service unavailability [10]	Client-based redundancy [24], dispersion and distribution [40]	Capacity, performance management overhead
Privacy	Control degree of data management	Unauthorized access and mining [31]	Client-based control schemes and encryption [33]	Management and performance overhead
Legal	Control degree of data locations	Provider out of market, and change of jurisdiction [36]	Global scale resilience in clouds [26], Federation and Dispersion [26]	Management and performance overhead

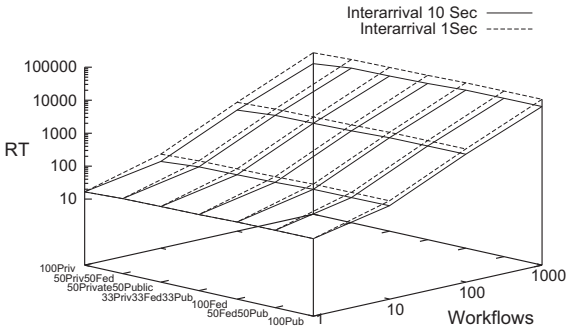

Fig. 17. The average response time produced when simulating the queuing of publications and workflows per diversification policy for various inter-arrival times.

Fig. 17 shows the side effects of the Pub/Sub queuing, for different diversification policies (*x* Axis), the side effects of the Pub/Sub queuing patterns on the response time of UPA App (*y* Axis) and different publications and workflows rates (*z* Axis). We configured the simulator for managing the queue size with a limit of 100 requests. The simulator determined the number of Pub/Sub patterns per queue before suffering an overload, which allowed SkyCDS to regulate the load on the producer and end-user sides. The queue size was updated to 10,000 requests for achieving results from the simulation of 1000 Pub/Sub patterns without producing overload. We simulated an inter-arrival of 10 Sec as it is the mean service time for all evaluated policies. A one second inter-arrival was also tested for producing saturation and to show the exponential growth of delay during the queuing of Pub/Sub patterns and workflows.

Fig. 18 shows the results corresponding to the UPA App running on the end-user side.

7.4. Case study: satellite images delivery service

Finally, we evaluated the performance of the SkyCDS as a whole service. In order to perform such an evaluation, we conducted a case study based on data from the European Space Astronomy Center (ESAC), located at Villafranca del Castillo

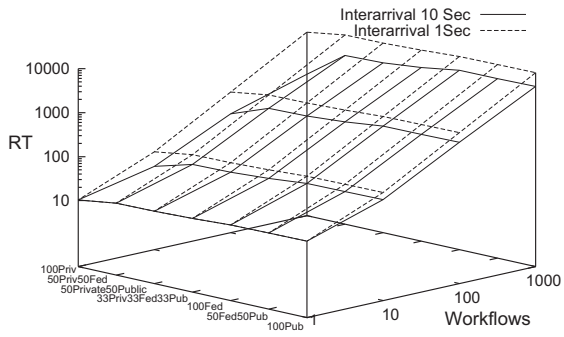

Fig. 18. The average response time produced when simulating the queuing of subscriptions and workflows per diversification policy for various inter-arrival times.

(Spain), which is in charge of the Soil Moisture Ocean Salinity or SMOS mission from 2009. This mission has generated a vast amount of contents that will be used to create the first Earth global salinity map.

An organization (*UC3M-Cloud*) distributes images of the SMOS mission in FITS format with a mean size of 44 MB to the end-users through the Internet. The publisher sent the images in sequential manner by using one single catalog, which was published in the SkyCDS platform. A set of organizations (*UC3M-Colme* and *Cinvestav*) and a set of end-users, from two countries spanning two continents, get the contents by using SkyCDS client Apps. *Cinvestav* is partner of *UC3M-Cloud*, which is partner of (*UC3M-Colme*). *UC3M-Cloud* and (*UC3M-Colme*) are regional partners, while *UC3M-Cloud* and *Cinvestav* are remote partners.

In this case, we configured the client Apps on the publisher side to disperse contents with (A) (LG) labels, so the traffic is sent to the federation by SkyCDS. The backup and reliability are priorities for this type of organizations in a content delivery service as contents could be acquired by end-users in different periods of time.

The client Apps of the end-users were configured to randomly subscribe publish contents from their organization. Each organization published the catalog to its partner, which retains the publication of contents until SkyCDS moves the chunks to the agents located on the organizations subscribing for the catalog, which considerably reduce the response time and masking the latency overhead. The agents and clients characteristics are presented in Table 2.

The following three configurations were defined in this case study:

- *DCS*: In this configuration, the end-user can access the contents by using an online distributed cloud storage system or DCS that does not include any redundant information.
- *DCS-Mirroring*: We implemented DCS in *UC3M-Colme* and *UC3M-Cloud*. This version of *DCS* includes a fault-tolerant strategy based on simple mirroring; as a result, two replicas are sent to the federated storage. The organization can withstand the failure of one cloud storage site by using this configuration.
- *SkyCDS*: In this configuration, the publishers and end-users produce Pub/Sub patterns and SkyCDS workflows, which were configured with a combination $(n = 5, k = 3)$. This configuration allows SkyCDS to withstand either two failures of storage locations or one site failure (not simultaneous).

Public configurations were not considered in this study because of the labels of the catalog.

Fig. 19 shows the mean response times observed by publishers when they distribute contents by using the studied configurations. Each point in this graph represents the mean response time of the delivery of ten contents. This means that 120 contents, with a mean size of 44 MB, were sent to the multi-cloud storage platform by the content delivery configurations.

Fig. 19 also shows that the response times of *DCS* configuration are better than the *SkyCDS* configuration. Nevertheless, *SkyCDS* configuration allows organization masking the outage of one cloud site and the failure of two storage locations (not simultaneous), while *DCS* is just a backup, in which any failure means lost data.

DCS-Mirroring improves the reliability of *DCS* configurations but it introduces a considerable overhead in the response times. Moreover, this configuration sends the whole file, so privacy and legal problems could arise. The *SkyCDS* system distributes anonymized and encoded chunks, which means that the *SkyCDS* agent and client knows the locations of the chunks and only they can reconstruct the contents. In addition, *SkyCDS* only produces up 66.7% of redundancy overhead while this overhead for DCS mirroring is of 100%.

7.4.1. Evaluation of the failure masking techniques

In this study case, we simulated a outage of the *UC3M-Colme* site and we measured the costs of the three type of failure masking levels supported by SkyCDS.

The first masking type is the failure of a storage location, in this case the overhead for publishers was 0.7% as the workflow is the same in faulty as steady state.

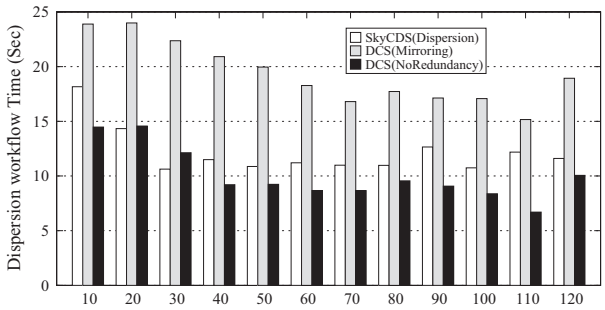

Fig. 19. The average response times observed by publishers when delivering 120 satellite images.

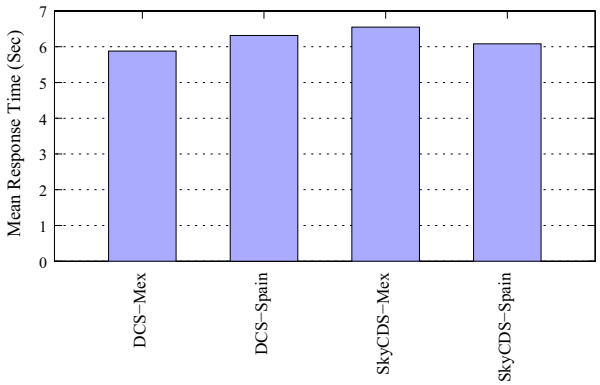

Fig. 20. The average response time per retrieved image observed by end-users.

In the one level, SkyCDS is masking *UC3M-Colme* site failure by retaining the publications for the end-users associated to *UC3M-Colme* organization. In this case, the publications were suspended but the workflows were accepted, the users of *UC3M-Colme* where distributed to its partner *UC3M-Cloud*, which only accepted subscriptions for published contents. The failure was transparent for Mexican end-users and introduced a controlled overhead for *UC3M-Colme* and *UC3M-Cloud* users.

The last level is the masking of latency effects. Fig. 20 shows the response time observed by end-users of Spain (regional partners) and Mexico (remote partners). Fig. 20 shows a mean response time of 6 s. This is possible because *UC3M-Cloud* delegates the publication authority of these contents for its end-users. *Cinvestav* retains the publication of each content until its agents retrieve all the chunks by using a retrieve workflow, the cost of this operation was 7 min in average, which was transparent for the Mexican end-users.

We also simulated the queuing of Pub/Sub patterns on the end-users Client App by using our simulation tool. The characterized metrics that were shown in Fig. 20 were used by this simulator to perform a set of experiments. Fig. 21 shows the Sub patterns queuing simulated. In this experiment, we simulated an inter-arrival of 6.3 s as it is the mean service time for each request produced by the tested policies. This also produces a FIFO queuing commonly used in archive and distribution segment of a space mission. An inter-arrival of 1 s was tested for producing saturation of the queuing of subscription patterns. Fig. 21 shows the exponential behavior observed in previously simulated experiments.

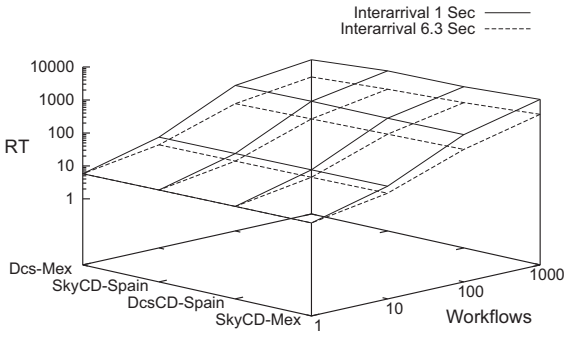

Fig. 21. The average response time when simulating the queuing of subscriptions and workflows on the end-user side.

As a result of the simulation of Pub/Sub patterns queuing that we performed, we concluded that the pressure on Pub/Sub queues can be reduced by deferring patterns for increasing the inter-arrival time and controlling the growth of Pub/Sub queuing on end-users and publishers side. This could be calculated in advance for improving the service experience of publishers and end-users in scenarios of high production and consume of contents. This makes sense to introduce a scheduling Pub/Sub task mechanism in the metadata manager as well as Client Apps, which also offering another level of failure masking.

8. Conclusions

In this paper we presented the design, implementation and performance evaluation of SkyCDS: a new resilient content delivery service based on a publish/subscribe overlay over diversified cloud storage. SkyCDS splits the content delivery into metadata and content storage flow layers. The metadata flow layer is based on publish–subscribe patterns for insourcing the metadata control back to content owner. The storage layer is based on dispersal information over multiple cloud locations with which organizations outsource content storage in a controlled manner. A new allocation strategy based on cloud storage diversification and failure masking mechanisms minimize side effects produced by temporary and permanent cloud-based services outages.

In SkyCDS, the content dispersion is performed on the publisher side and the content retrieving process on the end-user side (the subscriber), hence SkyCDS lowers the overhead of the content dispersion and retrieving processes by taking advantage of multi-core technology. In this collaborative scheme, the load on the organization side is reduced only to metadata management.

SkyCDS proposes a novel way to compare the storage/delivery options through risk assessment, which enables organizations to define road maps based on a matrix of risk ranking for cloud storage models.

We developed a SkyCDS prototype and we defined two evaluation scenarios for testing performance, metadata management overhead, and profitability of storage resource produced by SkyCDS. The first scenario was defined for evaluating the performance of SkyCDS on the publisher, end-users and organization side in each phase of the content delivery service (from encoding/decoding to data placement decisions). The second scenario was conducted as a study case based on information about satellite images of a space agency in which we also evaluated failures masking levels in a whole content delivery process. In this scenario, a set of organizations from two countries spanning two continents built a SkyCDS to deliver contents to end-users. We also evaluated the queuing of Publish/subscribe patterns by using a simulation tool based on characterized metrics taken from experimental evaluation.

The evaluation revealed the feasibility of SkyCDS in terms of performance, reliability and storage space profitability. It also revealed that the publishers take advantage of SkyCDS for ensuring a given content availability at a reasonable cost, a unified storage resource subsystem (ULS) for multi-cloud storage platform represents a cost-effective primary storage for SkyCDS and a secondary storage or backup for publishers and organizations. The evaluation also revealed that the organizations reduce workload of their servers to the control of the content management.

The results showed that the diversification policies improve the profitability of the cloud storage resources of the organizations and that the levels of failure masking enable end-users and publishers to get access of service even in temporary and permanent cloud-based services outages at a reasonable overhead.

The results taken from the simulation of Pub/Sub patterns queuing revealed that the pressure on Pub/Sub queues can be reduced by scheduling Pub patterns, which also reducing the growth of queues on client side.

9. Ongoing and future work

We are currently working on a load balancing system for achieving a trade-off between addressing the concerns of the content delivery users and the utilization of storage resources. We are also working on a dynamic scheme for adjusting the risk matrix assessment as it is based on heuristics by now. We will work on fine-tunning monitors for determining and launching as many SkyCDS agents as required to face up the increments of the content demands as well as site failures. We also will work on homomorphic tokens for ensuring communication of agents, clients and the SkyCDS manager with a strict control on the organization side. Finally, we are currently working on simulation of whole Pub/Sub patterns from publishers to end-users to define a Pub/Sub task scheduling for SkyCDS components.

Acknowledgment

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, Network for Sustainable Ultrascale Computing (NESUS).

References

[1] Google: Software Bug Caused Gmail Deletions <http://www.pcmag.com/article2/0,2817,2381168,00.asp> (posted 23.11.12, accessed 23.11.14).
[2] U.S. supreme court smit h v. maryland, 442 u.s. 735 (1979) 442 u.s. 735 smith v. maryland. certiorari to the court of appeals of maryland, 2010.

[3] Power Outage and Generator Failure Responsible for Instagram, Netflix Blackout, 2012 <http://www.itproportal.com/2012/07/04/power-outage-generator-failure-responsible-for-instagram-netflix-blackout/>.
[4] Uncovers Dangers of Cloud Storage Provider Lock-In <http://www.nasuni.com/news/52-nasuni_test_uncovers_dangers_of_cloud_storage/> (access 09.02.15).
[5] Amazon Cloudfront <http://aws.amazon.com/es/cloudfront/> (access 09.11.14).
[6] Amazon Simple Storage Service <http://aws.amazon.com/s3> (access 09.11.14).
[7] The Federated CDN for Service Providers <http://onapp.com/platform/onapp-cdn/> (access 09.11.14).
[8] Libcurl: The Multiprotocol File Transfer Library <http://curl.haxx.se/libcurl> (access 09.11.14).
[9] Smos: Soil Moisture and Ocean Salinit <http://www.esa.int/esaLP/LPsmos.html> (access 09.11.14).
[10] Nytimes: Amazon Cloud Failure Takes Down Web Sites, April 2011 <http://bits.blogs.nytimes.com/2011/04/21/amazon-cloud-failure-takes-down-web-sites/>.
[11] Emc Shuts Down Online Cloud Storage Service <http://www.storagenewsletter.com/rubriques/business-others/emc-stops-atmos-online-services/> (posted 06.07.10).
[12] It's Official, The Nirvanix Cloud Storage Service is Shutting Down <http://techcrunch.com/2013/09/27/its-official-the-nirvanix-cloud-storage-service-is-shutting-down/> (posted 27.09.13 by Alex Williams).
[13] H. Abu-Libdeh, L. Princehouse, H. Weatherspoon, Racs: a case for cloud storage diversity, in: SoCC, ACM, 2010, pp. 229–240.
[14] J. Behnke, T.H. Watts, B. Kobler, D. Lowe, S. Fox, R. Meyer. Eosdis petabyte archives: tenth anniversary, in: Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies, 2005, 2005, pp. 81–93.
[15] T.J. Bittman, L. Leong, Worldwide archival storage solutions 2011–2015 forecast: archiving needs thrive in an information-thirsty world, IDC. Market Analysis, October 2011, pp. 1–21.
[16] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina, Controlling data in the cloud: outsourcing computation without outsourcing control, in: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, 2009, pp. 85–90.
[17] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, Globally distributed content delivery, Internet Comput., IEEE 6 (5) (2002) 50–58.
[18] F.R. Duro, J.G. Blas, D. Higuero, O. Perez, J. Carretero, Cosmic: a hierarchical cloudlet-based storage architecture for mobile clouds, Simul. Model. Pract. Theory 50 (0) (2015) 3–19.
[19] M.J. Freedman, E. Freudenthal, D. Mazieres, Democratizing content publication with coral, in: NSDI, vol. 4, 2004, pp. 18.
[20] J. Gantz, D. Reinse, Extracting value from chaos, in: IDC-EMC2 Report, July 2011.
[21] J. Gantz, D. Reinsel, The expanding digital universe: a digital universe decade. are you ready? EMC Corporation, May 2010.
[22] A. Ghuloum, T. Smith, G. Wu, X. Zhou, J. Fang, P. Guo, B. So, M. Rajagopalan, Y. Chen, B. Chen, Future-proof data parallel algorithms and software on intel multi-core architecture, Intel Technol. J. 11 (2007).
[23] K. Gkoutioudi, H. Karatza, Task cluster scheduling in a grid system, in: Simulation Modelling Practice and Theory, Elsevier, 2010, pp. 1242–1252.
[24] J.L. Gonzalez, T. Cortes, Distributing orthogonal redundancy on adaptive disk arrays, in: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Part I on On the Move to Meaningful Internet Systems, Springer-Verlag, 2008, pp. 914–931.
[25] J.L. Gonzalez, R. Marcelin-Jimenez, Phoenix: fault tolerant distributed web storage, J. Converg., Section C: Web Multim. 2(1) (2011).
[26] J.L. Gonzalez, J.C. Perez, V. Sosa-Sosa, J.F. Rodriguez Cardoso, R. Marcelin-Jimenez, An approach for constructing private storage services as a unified fault-tolerant system, J. Syst. Softw. 86 (7) (2013) 1907–1922.
[27] G. Laatikainen, O. Mazhelis, P. Tyrväinen, Role of acquisition intervals in private and public cloud storage costs, Dec. Supp. Syst. 57 (2014) 320–330.
[28] E. Network, V.S. Information Security Agency (ENISA), Cloud Computing: Benefits, Risks and Recommendations for Information Security, Technical report.
[29] S. Perez-Ortiz, V. Sosa-Sosa, J.L. Gonzalez, L.M. Sanchez, J. Carretero-Perez, Taking advantage of federated cloud storage and multi-core technology in content delivery, in: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC '13, 2013, pp. 435–440.
[30] G. Pierre, M.V. Steen, Globule: a collaborative content delivery network, IEEE Commun., 2006.
[31] C.C. Porter, De-identified data and third party data mining: the risk of re- identification of personal information.
[32] G.T.R. Esposito, P. Mastroserio. Tucp008, standard ftp and gridftp protocols for international data transfer in pamela satellite space experiment, March 24–28 2003.
[33] M.O. Rabin, Efficient dispersal of information for security, load balancing, and fault tolerance, J. ACM 36 (2) (1989) 335–348.
[34] R. Rajan, H. Servaes, L. Zingales, The Cost of Diversity: The Diversification Discount and Inefficient Investment, Technical Report 6368, 1998.
[35] E.C. Reed, N. Chen, R.E. Johnson, Expressing pipeline parallelism using TBB constructs: a case study on what works and what doesn't, in: Proceedings of the Compilation of SPLASH '11 Workshops, ACM, 2011, pp. 133–138.
[36] E. Salveggio, Your (Un)reasonable Expectations for Privacy, in: Ubiquity, ACM, 2004.
[37] S. Singh, T. Jangwal, Cost breakdown of public cloud computing and private cloud computing and security issues, Int. J. Comp. Sci. Inf. Technol. (IJCSIT) 4(2) (2012) 17–31.
[38] R.H. Sloan, R. Warner, Unauthorized Access: The Crisis in Online Privacy and Security, first ed., CRC Press Inc., Boca Raton, FL, USA, 2013.
[39] J. Spillner, G. Bombach, S. Matthischke, J. Muller, R. Tzschichholz, A. Schill, Information dispersion over redundant arrays of optimal cloud storage for desktop users, in: Fourth IEEE International Conference on Utility and Cloud Computing, 2011, pp. 1–8.
[40] J. Spillner, J. Müller, A. Schill, Creating optimal cloud storage systems, Future Gener. Comput. Syst. 29 (4), 1062–1072.
[41] I. Technology Security Techniques Information Security Risk Management, ISO/IEC 27005:2008, Technical report.
[42] E. Walker, W. Brisken, J. Romney, To lease or not to lease from storage clouds, Computer 43 (4) (2010) 44–50.
[43] M. Wittmann, G. Hager, Optimizing CCNUMA locality for task-parallel execution under OPENMP and TBB on multicore-based systems, CoRR, abs/1101.0093, 2011.
[44] H.-S. Yeo, X.-S. Phang, H.-J. Lee, H. Lim, Leveraging client-side storage techniques for enhanced use of multiple consumer cloud storage services on resource-constrained mobile devices, J. Netw. Comp. Appl. 43 (2014) 142–156.