Boosting Content Delivery with BitTorrent in Online Cloud Storage Services

Rahma Chaabouni[*], Pedro García-López[*], Marc Sánchez-Artigas[*], Sandra Ferrer-Celma[*] and Carlos Cebrian[§]
[*]Universitat Rovira i Virgili, Tarragona (Spain), [§]TISSAT (Spain)
{rahma.chaabouni|pedro.garcia|marc.sanchez|sandra.ferrer}@urv.cat, ccbrian@tissat.es

I. Introduction

In classic storage services, the transfer protocol used is usually HTTP. This means that all download requests are handled by a central server which sends the requested files in a single stream. But, such transfer is limited by the narrowest network condition along the way, or by the server being overloaded by requests from many clients.

In this context, a number of studies have tried to combine BitTorrent content distribution technologies with Cloud environments. In fact, the efficiency of the BitTorrent protocol makes it especially suitable for massive content distribution while reducing bandwidth costs in the Cloud. Many previous works have focused on reducing download times for large contents using BitTorrent in Cloud settings. Either for bulk synchronous content distribution [1] or for reducing transfer times for cloud virtual images [2], [3], [4], BitTorrent proved to be a very efficient technology that outperforms classical centralized transfer solutions.

As we will explain in the next section, our approach is novel because it provides transparent and adaptive mechanisms to switch between traditional HTTP and BitTorrent technologies in Cloud Storage environments. None of the previous approaches targets this transparency or adaptivity.

In order to validate this new approach, we have modified an implementation of a personal Cloud system based on OpenStack Swift to accommodate BitTorrent. We developed also a message exchange system that captures the messages sent from the system and forwards them to a monitoring tool that gives real time information about the status of the transfers. Finally, we will show, during the demonstration and using our monitoring tool, how our adaptive policies can efficiently reduce transfer costs inside the Cloud.

II. Architecture

A. System Overview

Our storage service extends the classic model with enhanced techniques for content sharing. We use an implementation of an open source Personal Cloud system called *StackSync* which provides storage, syncing and sharing capabilities on top of OpenStack Swift. All the *StackSync* clients are connected to an OpenStack server in which their files are stored after being divided into small entities (called pieces or chunks).

As detailed in Fig. 1, to download a file, the *StackSync* client (1) sends an HTTP GET request to the Cloud Server. The latter verifies the existence of the file in the storage services and its "popularity". If there are few users requesting the file (the total number of requesters is inferior to a pre-defined threshold), the Cloud Server proceeds with the file transfer using the HTTP protocol. Otherwise, if the file is popular, the Cloud Server

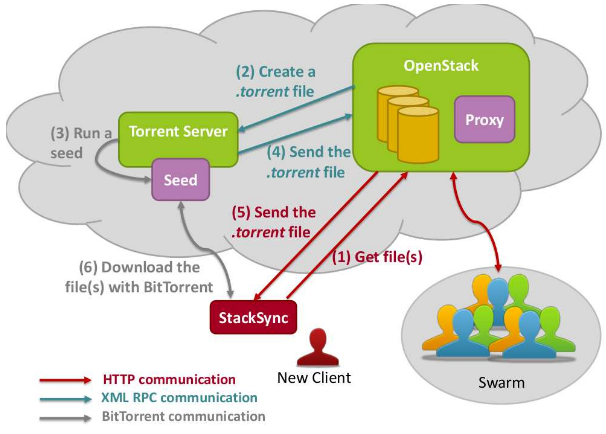

Fig. 1. Download scenario and the switch to BitTorrent

will decide to switch to the BitTorrent protocol. To do so, it transparently (2) commands the Torrent Server to create a torrent meta-data file and (3) run a corresponding seed. After doing so, the Torrent Server (4) sends to the Cloud the *.torrent* file created that will be (5) transmitted later to the client. The client, unaware of all these interactions, will then (6) start downloading the file using the BitTorrent protocol (from the Cloud Seed and/or from the other clients).

Evidently, the "older" clients who arrived before the switch to BitTorrent will also benefit from the switch if they did not finish the download. In fact, when a client requests a new chunk to be downloaded, he will realise that the downloading protocol has changed and will automatically adapt to the new one. Thus, each HTTP client will join the swarm with the pieces he already has, which means that he will be probably contributing to the swarm as soon as he switches to BitTorrent in a very transparent way.

B. Monitoring Tool

In order to validate the above-described architecture, we developed a special tool that can monitor the traffic between the components of the system and give real time information about the status of the transfers. Our tool offers the following features:

- Real-time measurements of different events and physical parameters of the system;
- Message queueing service between the different parts of the system and the graphic components;
- Dynamic charts to visualize the download progress.

These features are further detailed as follows:

1) Real-time measurements: To assess the performance of our model, we provide mechanisms for real-time measurement of the parameters of our system. These parameters can be physical (available bandwidth, consumed bandwidth...) or they can be in the form of execution events (a peer joining or leaving the swarm, a peer sending a piece to another peer...).

2) Message queueing system: To capture the previously stated parameters, we set up different sensors for each component of the system (see Fig. 2). These sensors are set to detect

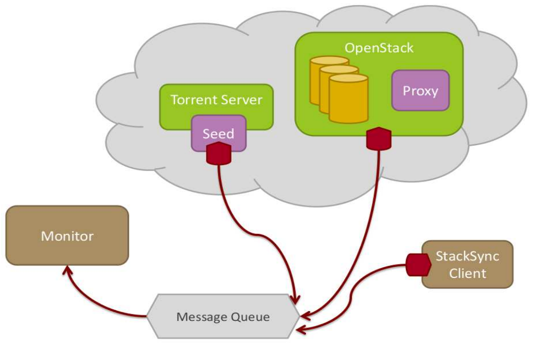

Fig. 2. Architecture with the sensors and the monitor

the "important" events. Depending on these events, the system sends the corresponding messages to the message queueing system. The messages are stored in a queue and consumed by the monitor which is responsible for updating the related graphical components.

3) Dynamic charts: Different statistical charts are included in our monitor in order to give us a real time projection of all the execution states. These charts are used to measure the evolution of different parameters both on the Cloud's and the peer's sides.

a) Cloud-Related Charts: In order to measure the contribution of our model in offloading the Cloud's serving, we include a line chart representing the evolution of the cumulative number of clients and the number of chunk-requests addressed to the Cloud over time. This chart is used to show that with increasing number of clients, the number of chunks requested from the Cloud is decreasing as the clients will be exchanging chunks among one-another. We measure also the contribution of an eventual local cache system integrated within the BitTorrent server. The cache's role is to save the chunks already requested from OpenStack Swift's storage nodes in a local memory to avoid repeated requests of the same chunk. We measure the percentage of the requests addressed to the cache in comparison with those from the the OpenStack Swift's storage nodes and we plot it in a dynamic pie-like chart.

b) The Network's Topology Graph: To be able to track the evolution of the transfers in the system, our monitor offers a real-time view of all the peers in the network in the form of a directed graph whose vertices are the nodes of the system and whose edges represent transfers in the form of arrows directed from the uploader to the downloader.

c) Global Charts: In order to measure the contribution of each component in the download process, we represent also the evolution of the all the chunks' requests sent through the system depending on their nature: HTTP or BitTorrent, and in the BitTorrent's case whether they are addressed to the Cloud seed or are the result of peers exchange.

III. Demonstration

We will deploy our Cloud in a rack at the university and during the demonstration, we will launch several StackSync clients. In fact, our monitor allows us to instrument these clients remotely and determine their entrance pattern. We will conduct our demo by making these clients request the same file at the same time in a flash-crowd scenario.
Using our monitoring tool, we will be able to follow the download evolution in real time and understand what is happening in the different components of the architecture so that we can change and adapt them to our goals. We will capture the shift between the download protocols when the threshold is reached, measure the contribution of the peers and calculate the improvements comparing to the classic download scenario. Moreover, this tool will help us determine precisely the best and most efficient threshold for different entrance patterns.

Acknowledgements

This work has been partially funded by the EU in the context of the project *CloudSpaces*: Open Service Platform for the Next Generation of Personal clouds (FP7-317555) and by the Spanish Ministry of Science and Innovation through the *RealCloud* project (IPT-2011-1232-430000).

References

[1] S. Priyanka, R. Kalpana, and M. Hemalatha, "Reducing upload and Download Time on Cloud using Content Distribution Algorithm," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 1, pp. 101–105, 2013.

[2] R. Wartel, T. Cass, B. Moreira, E. Roche, M. Guijarro, S. Goasguen, and U. Schwickerath, "Image distribution mechanisms in large scale cloud providers," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 112–117.

[3] M. Schmidt, N. Fallenbeck, M. Smith, and B. Freisleben, "Efficient distribution of virtual machines for cloud computing," in *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, 2010, pp. 567–574.

[4] J. Reich, O. Laadan, E. Brosh, A. Sherman, V. Misra, J. Nieh, and D. Rubenstein, "VMtorrent: virtual appliances on-demand," in *SIGCOMM*, 2010, pp. 453–454.