Simulator Of A "Weather" Cloud
Ksenia Khramenkova, Olivier Hermant, Renaud Pawlak

To cite this version:
Ksenia Khramenkova, Olivier Hermant, Renaud Pawlak. Simulator Of A "Weather" Cloud. 10th Finnish-Russian University Cooperation in Telecommunication Workshop (FRUCT), Apr 2012, St.-Petersburg, Russia. <hal-00909649>

HAL Id: hal-00909649
https://hal-mines-paristech.archives-ouvertes.fr/hal-00909649
Submitted on 26 Nov 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Simulator Of A “Weather” Cloud

Khramenkova Ksenia
State University of Aerospace Instrumentation
190000, Saint-Petersburg, Bolshaya Morskaya, 67, Russia
ksu.khramenkova@gmail.com

Hermant Olivier and Pawlak Renaud
Institut Supérieur d'Electronique de Paris
75006, Paris, rue d’Assas, 21, France
{olivier.hermant, renaud.pawlak}@isep.fr

Abstract

In this article a cloud simulator for the “weather” cloud is considered. The purpose of such a simulator is evaluating different cloud architectures and algorithms before implementation. The main idea is to analyze the performance beforehand, in order to avoid unsuitable algorithms being implemented in a real cloud. Two methods of request allocation policies to the nodes are considered. Their behavior in terms of interaction with nodes’ cache-memory is compared. Finally, advantages and issues of both algorithms are highlighted.

Index Terms: cloud computing, modeling, algorithms, cache

I. INTRODUCTION

“Cloud” computing is a relatively recent term, built on decades of research in virtualization, distributed computing, utility computing, and more recently networking, web and software services. It implies a service oriented architecture, reduced information technology overhead for the end-user, great flexibility, reduced total cost of ownership, on-demand services, etc [1].

Cloud computing delivers infrastructure, platform, and software as services, which are made available as subscription-based services in a pay-as-you-go model to consumers. These services in industry are respectively referred to as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

Cloud computing can be defined as “a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers” [8]. Some examples of emerging cloud computing infrastructures are Microsoft Azure [2], Amazon EC2 [3], Google App Engine [4], and Aneka [5] [6].

In this paper the creation of a cloud simulator and the dispatching of tasks on nodes are considered. An algorithm of sending requests by the controller to the nodes is implemented so that the largest possible part of the requests is already in the chosen node's cache memory. The goal of this task management algorithm is to reduce the processing time of requests.

In the first section we talk about terms of cloud computing. In the second section a cloud architecture, running the service we aim to model, is described. The cloud simulator is
introduced in the third section. And cache algorithms comparison results on the real traffic are presented in section IV. The paper ends with some conclusion remarks.

II. CLOUD DESCRIPTION

CAP 2020 is a French company that aims at providing various services for the French agriculture, especially services that help the farmers to analyze agricultural data in their fields in order to perform more efficient and environmental-friendly farming.

For cost reasons, most farmers use machine to machine platforms such as european-wide weather services like Meteo France and Numtech. However, the precision of such continental services is not sufficient for the farmers who need to work at very small scale to optimize their production (1km x 1km areas). Thus, farmers also use specific sensor networks such as PESSL (weather) or Agriscope (weather and agriculture specific sensors). Examples of such sensors can be temperature, soil or air humidity sensors, soil acidity, and so on.

The aim of CAP 2020 is to gather all those heterogeneous sources of existing data and to integrate the data coming from all these sources with interpolation in order to provide homogeneous services to the farmers, as close as possible to their fields. The weather service provided by CAP 2020 allows the farmers to define Virtual Weather Stations (VWS) in their fields for accessing weather history and forecast at a very precise point. The provided data are calculated by interpolation from weather data grids (a set of points and associated data).

The weather data grids' data come both from the institutional weather services, and from many local sensor data. Those last data are integrated to the grids through a global interpolation algorithm implemented and hosted within a dedicated server by Estimages, a company specialized in data interpolation and imaging for oil company.

Thus, in order to know the value of a given data for a given VWS, the service provided by CAP2020 takes the available grids, including Estimages generated grids, and performs a local interpolation operation. There is three kind of weather data grid precision, that split France into 15km x 15km, 5km x 5km and 1km x 1km squares. A user may, or may not, have access to a given precision, depending on its rights and on the data type.

Due to the extremely large amount of data (we typically receive 20 million lines of data (date/time type unit value, where the type can be temperature, humidity, wind speed or direction and solar radiation received) each day for a 5km x 5km weather data grid, that contains 68272 points), and to the frequency and complexity of requests (it is projected that the daily amount of requests will soon reach 1,4 million), the service is deployed on a cloud computing architecture [7]. Each node of the cloud replicates the data that are split into several files, depending on the region and on the date/time the data were collected. When asked for, the service on the node performs the interpolation and sends back to the controller the interpolated results.

Very soon after its deployment, we noticed that the computation time required by the same request may vary drastically: it can be up to 4 times faster in some cases. An analysis of this phenomenon lead us to remark that requests are processed faster when the data files required by the interpolation algorithm are already in the cache memory of the system. Indeed, a huge
amount of CPU time is wasted when transferring big files from the hard-disk drive to the system's RAM memory, before beginning the interpolation operations on the given VWS.

That is why an optimization of the task allocation process is crucial. A better algorithm results in a gain of time, and thus allows the CAP2020 service to decrease the resources usage. The goal we identified for this algorithm, is to send requests on the node that, most probably, has already loaded the required data files in its cache memory.

Unfortunately, we cannot do experiments directly on the CAP2020 cloud architecture since this would overload and threaten the stability and availability of the provided service. This is why we develop the simulation of such a cloud architecture, so as to make experiments on the task allocation algorithm and to choose the best one.

III. WEATHER CLOUD SIMULATOR

A cloud simulator is used for modeling cloud architecture for performance analysis and optimization. During a simulation bugs can be discovered and fixed before moving on to the implementation phase in which debugging will be a lot of work.

Figure 1 presents the schema of the cloud simulator architecture. The controller is connected to each node via the channels. The bandwidth of each channel is adapted dynamically depending on the number of active channels, so as to be closer of a real network. Each node contains resources like CPU power, memory and cache (Cache algorithm in nodes is the Last Recently Used algorithm) and provides a number of services which uses the node's resources in a predefined way. Modeling the cache memory in nodes is of crucial importance since it has been noticed, in the Cloud Description of section II [7] that requests on files already in cache memory takes up to four times less, which must be reflected in this simulator.

![Fig. 1 Schema of cloud simulator](image-url)
The controller receives requests from users. After the controller by task allocation algorithm sends them to the nodes. Each node processes the request and performs the specified service. The node then issues a response, which is a result of that service like computation results or storage, back to the controller. The controller sends the response to the user. This is done using a discrete synchronized time, step by step. In particular, we aim to measure the "logical time" taken by a request to be processed.

There are several auxiliary modules in the cloud simulator such as:
- Services which are provided by nodes.
- Requests which demand the node to perform a specific service.
- Responses which are the result to the specific service. They are sent in return to the controller.

The cloud controller aims to store nodes and requests statistics, send requests to chosen nodes (via the channel) and get response from nodes (via channel).

A request is characterized by its service type, its size and the list of documents it needs from the hard disk drive, that are supposed to be identical in each node. For the simulation, we chose to have 1000 different documents, that may be present in the cache memory of a node, or not. The size of the cache memory of each node is 250.

Each service implements three methods which, depending on the request's size, evaluate the memory and the CPU work required and calculate the output data size. The CPU work required may depend on the percentage of cache hit as in the simulation of the Weather Cloud of Section II, or may not. It also depends on the input data size in different manner, for instance for a naive sorting algorithm, it is quadratic on the input size.

Two different task management algorithms have been implemented on the controller:
- Round Robin (RR) algorithm does not analyze the request. The controller selects a node pointed to by a circular counter from a list, after which the counter is incremented, modulo the total number of nodes. This algorithm does not need to know anything about nodes or its cache-memory tables.
- The algorithm with an analysis of the contents of the request (AAR) analyzes the content of the request namely the array of documents requested by user. The algorithm assumes that documents between 0 and 249 are in the cache memory of node 1, between 250 and 499 – in the cache memory of node 2, between 500 and 749 – in the cache memory of node 3 and between 750 and 999 – in the cache memory of node 4.

The controller receives and analyzes the percentage of requested documents (cache hit) for each node and chooses by the biggest value where the current request has to go. Thus cache hit have to grow and time required for processing have to decrease.

IV. Simulation Results

As a result of the modeling we get a table containing: task identifier, service type, node identifier, start time, finish time, cache hit percentage, input data size, output data size.
The values of the last two parameters can vary. They depend on the called service and of the chosen size of the input. The task identifier is the number of the request. This parameter is important for analyzing dense stream of requests. Because in a cloud can be low-power nodes, which slowly process requests, there is a parameter node identifier. It is needed to check each node in the cloud and watch them during the entire simulation. The start time describes time when request entered node. Finish time describes the time when the response reaches controller. If work was with cache in table there is the percentage of cache hit.

The comparisons are done for RR and AAR algorithms; array of documents in request is filled randomly. The service analyzed is a simulation of the Weather Service discussed in Section II that uses the cache memory. So as not to depend on the initial state of the cache memory (empty at the beginning), each test consist of a large amount of requests (from 1000 to 10000), each time with a fixed size of the document array.

The figure 2 shows processing request time against the amount of requested documents. With increasing volume requested documents the processing time using the AAR grows, but the processing time using the RR algorithm decrease. If the amount of requested documents is more than 100 per request, the idea to analyze requests fails, because processing time by node is the same.

![Fig. 2 The processing request time](image)

The percentage of cache hit decreases when the amount of requested documents grows (see figure 3) and therefore, the processing time increases. When the number of requested documents is 100 per request, the work of AAR is equal to the work of the RR algorithm. So it does not matter what algorithm has been chosen in the controller to distribute requests.
These results deserve some comments. First of all, the array of documents has been chosen randomly. So that every time we send a request to a node, its cache memory is filled with random documents. In the AAR case, more than 25% of the documents are in the target range, but the cache memory of the node is polluted with (at most) 75% of the other documents. This subsequently decreases the cache hit of the next requests. Better results would have been obtained if we had chosen document number in a row, which sometimes happens in the CAP2020 service (when a range of dates is selected, for example "last week", or "last month"). The first document's number $n$ is still random, but then we select documents $n+1 \ldots n+m$. Then, at least 50% of the documents will be in the chosen node's range. This has to be implemented yet.

Second, we can observe in the RR case of figure 2 a decrease of the processing time, while we would expect a stable time. This can be explained by the fact that the cache memory is empty at the beginning, and the more documents there is in a request, the faster it fills, and the faster we obtain a stable 25% cache hit.

V. CONCLUSION

This article discusses the simulation of a cloud that contains information about weather in France. Comparison was done for 2 algorithms: for Round Robin algorithm and for the algorithm with an analysis of the contents of the request. The results of the analysis are: the second algorithm gives a small gain in time and cache hit on small amount of requested data. So, we need to improve the algorithm with an analysis of the contents of the request. When requests are distributed by controller, it has to know dynamically the state of the cache memory of the nodes so as to calculate dynamically the expected cache hit percentage. After analyzing array of
requested documents in the current request, the controller can then choose the optimal node for the current request, from the point of view of cache hit.

**REFERENCES**


