Preliminary version. Final version appears in *Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid'12),* May 13-16, Ottawa Canada

A Model and Decision Procedure for Data Storage in Cloud Computing

Arkaitz Ruiz-Alvarez, Marty Humphrey
Computer Science Department
University of Virginia
Charlottesville, VA, USA

***Abstract*—Cloud computing offers many possibilities for prospective users; there are however many different storage and compute services to choose from between all the cloud providers and their multiple datacenters. In this paper we focus on the problem of selecting the best storage services according to the application's requirements and the user's priorities. In previous work we described a capability based matching process that filters out any service that does not meet the requirements specified by the user. In this paper we introduce a mathematical model that takes this output lists of compatible storage services and constructs an integer linear programming problem. This ILP problem takes into account storage and compute cost as well as performance characteristics like latency, bandwidth, and job turnaround time; a solution to the problem yields an optimal assignment of datasets to storage services and of application runs to compute services. We show that with modern ILP solvers a reasonably sized problem can be solved in one second; even with an order of magnitude increase in cloud providers, number of datacenters, or storage services the problem instances can be solved under a minute. We finish our paper with two use cases, BLAST and MODIS. For MODIS our recommended data allocation leverages both cloud and local resources; it incurs in half the cost of a pure cloud solution and the job turnaround time is 52% faster compared to a pure local solution.**

Keywords: data allocation; cloud computing; integer linear programming

I. INTRODUCTION

To decide which cloud platform [1] is best for a new cloud application, typically a designer focuses on compute capabilities as the deciding factor. That is, after it is decided if it will be public-, private-, or hybrid-cloud, a high-level decision is made regarding PaaS vs. IaaS, and then a subsequent decision selects the particular platform within the class (e.g., IaaS and then Amazon EC2). Implicit in this process is a belief that the storage capabilities of each cloud are basically equivalent or at least not sufficiently distinguishable to warrant closer consideration as to the choice of cloud platform.

However, we believe that data capabilities should be a first-class consideration when selecting a cloud platform, at the same level of importance as computation. Arguably computation is more flexible: a Windows application can run on a native Windows OS (local Windows HPC cluster or Windows Azure) or within a virtual machine (local Eucalyptus cluster or Amazon EC2). Storage services, on the other hand, present many different options whose capabilities are sometimes exclusive to a cloud provider; choices range from traditional files (NFS) and SQL databases in local clusters to a variety of cloud services (for Amazon there are S3, EBS, SimpleDB, RDS and ElastiCache).

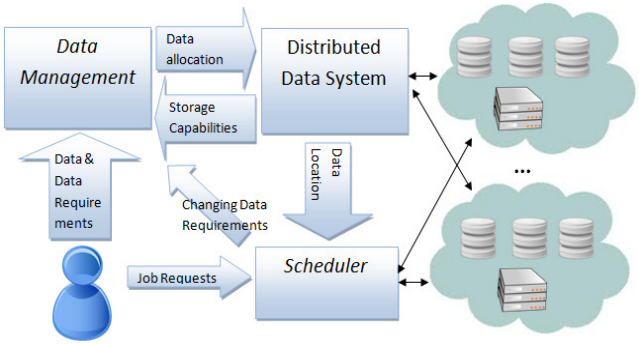

Figure 1. Overview of our system. In this paper we focus on the Data Management component.

In previous work [2] we presented the first phase of a system designed to help with the cloud storage service selection decision. First, we developed an XML schema that describes the capabilities (both functional and non-functional) of the storage services of Amazon, Windows Azure and local clusters. We can express different attributes like the encryption capability of S3, the performance of Windows Azure Tables for a read operation under multiple concurrent clients or the capacity of the local NFS file system. Second, we encoded different user requirements; each requirement will look at a certain storage service and decide whether there is a match (true/false decision). Custom user requirements can be added easily, some examples of the ones developed by us are: "data must be located within the United States", "data must be triple replicated or with specified durability of >99.999%" or "data can be accessed concurrently". The input for our prototype application is a description of the different datasets that the application uses, together with their storage requirements. The output is, for each dataset, a list of storage services that meet those requirements, along with cost and performance estimates.

For example, an application may use three different datasets: several GBs of input satellite data, intermediate storage shared among different processes and several MBs of output files. Our prototype will give a list of possible storage services for dataset; for the satellite data it may be: Amazon S3 in Virginia and California, Azure Blobs in South-central and North-central US, etc. Similar lists are presented for the other two datasets. The user would then choose an allocation from

these options: satellite input to S3, intermediate data to SimpleDB and output data to S3 RRS in Amazon Virginia.

While we believe that this prototype is valuable, there are three important limitations: the list of storage options for each dataset is not ordered, that is, we do not present a preferred storage option; each dataset is analyzed in isolation so it is not obvious what the best global solution is; and the computational side (number of application runs, cost per hour, machine speed, etc.) is not taken into account. In this paper we seek to address these limitations so we can produce a global data allocation solution that balances cost and performance of both storage and computation. If the best storage service for a single dataset resides in a cloud that does not have good choices for the rest of the application data, then we may arrive to a sub-optimal allocation of data. As we will show, trying to find an optimal solution increases the complexity of the problem to NP hard. We provide a model for this data allocation problem and a software implementation that is both fast and scalable.

Figure 1 shows a general overview of our system. On the right side we have the different datacenters with storage and compute services that cloud providers and others (local clusters) provide. On the left side we have the user that provides the data, the data requirements, and information about the execution of the application (number of runs, duration, datasets accessed). We have already mentioned our approach [2] to describe the *storage capabilities* and *data requirements* in a machine readable format and perform a capability-based matching. The output of this first stage is taken as the input for this paper; our final goal is to produce the *data allocation* decisions. Thus, we complete the component *Data Management* in Figure 1; we leave the *Scheduler* and *Distributed Data System* components as future work.

We will first present our problem model, whose solution represents the data allocation decision. Factors such as storage cost, compute cost, latency, and bandwidth are combined into an objective function that has to be minimized. The combination of this function and additional restrictions (linear constraints) forms an integer linear programming problem, which is NP hard. In order to solve this computationally hard problem we use a modern ILP solver (lp_solve [3]); we show that, for simple use cases, a couple hundred milliseconds are enough to solve the problem instance. For a more complex application with 3 datasets and where each dataset could be matched to 48 possible storage systems the solver takes 1 second to solve the ILP problem. We show that even with an order of magnitude increase in the number of possible storage and compute services our approach is able to come up with an optimal data allocation within seconds.

We present two use cases for our system: BLAST and MODIS. For BLAST we take our standard model and include one additional restriction: a monthly budget. Thus, the user may ask for the best data and compute allocation which fits her budget. Latency and bandwidth are not the only performance metrics that we have considered. For MODIS we add job turnaround time as another factor in our objective function. In this case we try to make the allocation decisions that will minimize job turnaround time the most, while still meeting the data requirements and the budget.

In summary, the contributions of this paper are:

- We introduce a formal model to represent the problem of allocating resources (storage and computation) to services offered by different cloud providers.
- We extend the implementation of our data management system with an ILP solver to provide a timely optimal solution to our data allocation problem.
- We show that other metrics (job turnaround time) and restrictions (monthly budget) can be included in our framework; we use the BLAST and MODIS applications as examples. For MODIS, our storage and compute allocation for a budget of $1,000 has an average turnaround time of 1.64 hours per job. Had the user selected a pure cloud allocation (Azure) the monthly cost would have been over $2,000. Conversely, a purely local solution would have been cheaper, but the turnaround time would have been 52% higher.

The rest of this paper is organized as follows: following this introduction we present the related work in Section 2. Section 3 introduces the formalization of the resource allocation problem. Section 4 shows the software implementation of this algorithm; here we present two examples, the performance of the algorithm, the scalability of our approach, and the sensitivity of the (optimal) solutions that the algorithm produces. In Section 5 we present the BLAST and MODIS use cases. Finally, we discuss the current limitations of our approach and outline future work in Section 6 before concluding with Section 7.

II. RELATED WORK

The theoretical foundation of the file allocation problem was formalized by Chu several decades ago [4]. In this problem we have a set of computers interconnected by a network which provides storage for a set of files. Each file can be stored in multiple computers, and the problem model takes into account the cost and capacity of storage and transmission, and the maximum latency allowed for each access. The optimal solution is the one that minimizes the cost of storage and transmission. Chu's work formulates the problem as a zero-one integer linear programming problem, which is NP hard. This problem model, however, does not address some of the users' requirements outside cost and maximum latency; it also does not take into account the possibility of multiple sites. Thus, it does not apply directly to data management in cloud computing. Other similar work by Casey has formulated a problem model for allocating multiple replicas of a file in a distributed system [5] taking into account the cost of storage and data transmission and the read and update queries: this problem is still NP hard. Subsequent work on the file allocation problem has addressed the complexity of these models by filtering sites that participate or not in an optimal solution [6], devising polynomial-time approximation algorithms based on a reduction of the file allocation problem to the Knapsack problem [7] or other heuristics that iteratively refine feasible initial solutions [8]. Dowdy and Foster [9] identified 12 different models of the file allocation problem which differ on several parameters: minimizing cost, execution time, access time, response time or maximizing throughput; considering single files, multiple files or data and program files; etc. Later

variations of the problem [10], [11] considered also a dynamic approach as the storage needs change over time and also the location of program files associated with data files [12]. To the best of our knowledge, the most common software for managing data grids (SRB, iRODS, GPFS, HPSS) does not implement these file allocation algorithms. In data grids we can usually find a dedicated part of a site to storage, and the rest of the nodes access data through the network. Thus, there is not a concern for optimizing data storage costs at the individual computer level and the majority of the access to the data occurs within the site. However, with the introduction of cloud computing there is the possibility of renting storage space at different sites, making the file allocation problem relevant again. The unique characteristics of our approach are: our problem model was built specifically for data management in cloud computing (as opposed to within a local cluster), and we present an implementation that is fast enough to provide an optimal solution. This fast solution is based upon recent advances on the development of efficient boolean satisfiability solvers [13] and ILP solvers [3].

III. Resource Allocation Problem Model

In this section we describe our mathematical model used to express the data allocation problem in cloud computing. Our goal is the following one: to select the best storage systems which meet the user's data requirements and optimize cost and/or access latency/bandwidth. Recall that the first stage in our system (described in Figure 1) is a matching process whose inputs are a list of user's requirements and the storage services' capabilities. The output of this first stage is a list of compatible storage services for each dataset in the application; these lists constitute the input for our data allocation problem. We use integer linear programming to model this problem. The general idea is to include the cost, latency, and bandwidth as parameters in the objective function that needs to be minimized. Of the variables that we introduce, 0-1 integer variables tell us which storage systems will store which datasets ($x_{i,j}$); the solution to the problem will be an optimal assignment of datasets to storage systems. We also introduce integer variables that represent the amount of computation required per month; the solution also yields an assignment of computation to cloud sites ($computation_k$). We use additional linear constraints to enforce different restrictions; for example, that each dataset is stored in at least one storage system and that each site can support the computations that access each dataset. A glossary of all the terms used in the equations in this section is shown in **Error! Reference source not found.**This table gives the type, description and source (user input, part of the storage capabilities information and part of the solution) for each variable. The objective function is the following one, where each w_i is the combination of a weight assigned by the user and a normalizing factor:

$$MIN(w_1 \times Average\ Storage\ Cost + w_2 \times Average\ Compute\ Cost + w_3 \times Average\ Latency + w_4 \times Average\ Bandwidth) \quad (1)$$

We need to combine every term in a meaningful way. In order to evaluate cost (in dollars), latency (milliseconds), and bandwidth (MB/s), we consider the average over all the application datasets and normalize it. We normalize each parameter to the average calculated from all the cloud storage systems (optionally the user may provide their own). The user will give us α_i, which represents the weight (between 0.0 and 1.0, totaling 1.0) for each term:

TABLE I. Problem Model Variables

Name	*Description*	*Source*
$X_{i,j}$	Binary, allocation of $dataset_i$ in $storage_j$	Solution
$Y_{i,j,k}$	Integer, number of data transfers for $dataset_i$ from $storage_j$ to $site_k$	Solution
$Computation_k$	Integer, number of application runs at $site_k$	Solution
Dataset $Size_i$	Float, size of $dataset_i$ in GB	User input
Dataset $Request_i$	Float, number of monthly storage requests for $dataset_i$	User input
Dataset $Usage_i$	Float, percentage of $dataset_i$ accessed by the average application run	User input
Computation Length	Float, number of hours per application run	User input
Site $Capacity_k$	Float, number of computational hours available per month at $site_k$	User input
CostTransfer $IN/OUT_{j,k}$	Float, cost in dollars for GB of data transfer IN/OUT $storage_j$ to $site_k$	Storage Capabilities
Cost $Requests_j$	Float, cost per request on $storage_j$	Storage Capabilities
Cost $Hour_k$	Float, cost per compute hour in $site_k$	Storage Capabilities
Cost $Storage_j$	Float, cost per GB per month in $storage_j$	Storage Capabilities
$Latency_{j,k}$	Float, latency in ms when accessing data in $storage_j$ from $site_k$	Storage Capabilities
$Bandwitdh_{j,k}$	Float, bandwidth in MB/sec when accessing data in $storage_j$ from $site_k$	Storage Capabilities

$$MIN\left(\frac{\alpha_1}{average\ cost\frac{GB}{month}} \times Average\ Storage\ Cost + \frac{\alpha_2}{average\ cost\ per\ hour} \times Average\ Compute\ Cost + \frac{\alpha_3}{average\ latency\ ms} \times Average\ Latency + \frac{\alpha_4}{average\ bandwidth\ MB/s} \times Average\ Bandwdith\right) \quad (2)$$

For example a solution could have a normalized storage cost of 1.15 and a normalized latency of 0.95, meaning that the storage cost is on average 15% more expensive but latency is 5% better. The best solution is then determined by the α_i. It is possible to easily give the user more control by expanding the formula and introducing $\alpha_{i,j}$ (weights that depend on each parameter and the dataset). This way it would be possible to fine tuned each dataset in case there is one that is critical to the application flow, which could have, for example, a low latency requirement.

We will first expand the Average Storage Cost term, which represents the average monthly cost per GB of data stored:

$$Average\ Storage\ Cost = \frac{Storage\ Cost + Transfer\ Cost + Request\ Cost}{Total\ Gigabytes\ Stored} \quad (3)$$

$$Storage\ Cost = \sum_{i,j} x_{i,j} \times datasetSize_i \times costStorage_j \quad (4)$$

$$Request\ cost = \sum_{i,j} x_{i,j} \times datasetRequests_i \times costRequest_j \quad (5)$$

$$Transfer\ Cost = \sum_{i,j,k} y_{i,j,k} \times datasetSize_i \times datasetUsage_i \times \left(costTransferIN_{j,k} + costTransferOUT_{j,k}\right) \quad (6)$$

In our actual implementation, these equations are more complex since the pricing structure for some cloud providers is not flat: there is layering pricing, monthly plans, special offers, etc. However, these equations reflect the most important parameters involved and we'll use them in our problem model description for simplicity. The calculation of transfers cost is straightforward if the dataset is not replicated across different storage systems. If we want to allow datasets to be stored in several storage systems then each computation is going to select one of the replicas based on the transfer cost and access latency/bandwidth; depending on the actual values for the α_i. In order to express this in the objective function we introduce the variable $y_{i,j,k}$ which represents the number of transfers of data (per month) of $dataset_i$ from $storage_j$ to $site_k$ (where presumably there are some computational resources that process the data). Together with this part of the objective function we need additional constraints for each variable $y_{i,j,k}$:

$$\begin{array}{cccc} \sum_j y_{1,j,1} = computation_1 & \cdots & \sum_j y_{1,j,K} = computation_K \\ \vdots\ I\ times & & K\ times & \\ \sum_j y_{I,j,1} = computation_1 & \cdots & \sum_j y_{I,j,K} = computation_K \end{array} \quad (7)$$

These constraints establish that, for every $dataset_i$ and for every $site_k$, the total number of $dataset_i$ transfers needed (from any $storage_j$, thus $\sum_j y_{1,j,1}$) equals the number of computations at that $site_k$. An additional restriction is that if we transfer data from $storage_j$ to $site_k$, the data must be there (if $x_{i,j}$ is zero, then so it is $y_{i,j,k}$):

$$y_{i,j,k} \leq x_{i,j} \times site\ capacity_k \quad \forall\, y_{i,j,k} \quad (8)$$

The second term, the compute cost equation, can be expressed as:

$$Average\ Compute\ Cost = \frac{\sum_k computation_k \times costHour_k}{Number\ of\ compute\ hours\ per\ month} \quad (9)$$

The third main term, *Average Latency*, can be expressed as:

$$Average\ Latency = \frac{\sum_{i,j,k} y_{i,j,k} \times latency_{j,k} \times datasetUsage_i}{Number\ of\ datasets \times Number\ of\ application\ runs} \quad (10)$$

This term leverages the variables $y_{i,j,k}$ introduced for the cost calculation. Here we find the latency for each data transfer, multiply by the weight of that data transfer and divide it by the number of data transfers so we can obtain the average access latency.

The expression for fourth term, *Average Bandwidth*, mirrors *Average Latency*:

$$Average\ Bandwidth = \frac{\sum_{i,j,k} y_{i,j,k} \times bandwidth_{j,k} \times datasetUsage_i}{Number\ of\ datasets \times Number\ of\ application\ runs} \quad (11)$$

Aside from the objective function, we must provide our integer linear programming solver with these additional linear constraints (data has to be stored somewhere, $computation_k$ does not exceed site capacity and all $computation_k$ add up to the application needs):

$$For\ each\ dataset_i: \sum_j x_{i,j} \geq 1 \quad (12)$$

$$For\ each\ site_k: computation_k \times computationLength < site\ capacity_k \quad (13)$$

$$Computation: \sum computation_k \geq Number\ of\ app.\ runs\ per\ month \quad (14)$$

In this section we have considered the following four factors: storage cost, compute cost, latency, and bandwidth. We believe that these are important metrics for the user; others are certainly possible. Some, such as availability or durability, come into play in our previous stage, where datasets are matched to possible storage systems based on capabilities and requirements. Thus, these metrics come into play as a filter, where the decision is binary and do not participate in the objective function. We present an example in Section V.B where a new metric (job turnaround time) should be included into the objective function so the solution strives to minimize this metric.

IV. Resource Allocation Problem Solver

In this section we describe the software implementation that solves the problem model introduced in the last section. As we have mentioned in the introduction, this software does not exist in isolation or as a purely theoretical approach; rather we see the data allocation problem solver as a component of a larger project, shown in Figure 1. The first component is the storage capability matcher, which takes two inputs: a machine readable description of the cloud storage systems (storage capabilities), and a set of requirements from the end user for each dataset in the application. The output is a filtered set of possible storage systems for each dataset, along with the information that our problem model requires. Like the storage capability matcher that it interacts with, our solver is implemented as a C# prototype that uses the Microsoft Solver Foundation [14] library. Thus, we have the ability of use the default solver or plug in another that is compatible with the Solver Foundation interface; for the experiments presented in this paper lp_solve [3] was selected as the ILP solver. At this stage, the output of our GUI is a textual representation of the problem instance and the solution, along with additional details/statistics that MSF provides; further automatization is planned (Section VI). The experiments have been run on our desktop machine, an AMD Athlon II X4 2.90GHz with 6 GB of RAM running Windows 7.

A. Basic Examples

Our first example is taken from our previous paper on storage service selection [15]. In this case, we present an application with three different sets of data: a) satellite data (10 GB), b) intermediate storage shared by workers (1 GB), and c) output files (2 GB). The solver presents us with the following optimal solution: satellite data goes to both Amazon S3 in Virginia and the local NFS cluster, both intermediate results and output go to the S3 Reduced Redundancy Storage, 30 of the application runs happen in the local cluster (this maxes out the allocated capacity for this application) and the rest of them go to Amazon EC2 in Virginia. This solution takes into account many issues: S3 is selected to comply with the user requirements of high durability for satellite data, a local copy of the input dataset is created to reduce transfer costs, local computational resources are used to minimize cost, and additional cloud resources are chosen based on cost and access

latency. The problem model for this example has a total of 149 variables and it takes 88 ms to solve. The problem input and the output from the solver for this example and the next one is available on our website [16].

Our second example is a MapReduce application which has an input dataset of 1 TB and generates 10 GB output. Local resources can support a normal daily run of this application, but a few times per month a more complex analysis is required. For this use case we get the following solution: store the input data in both the Amazon cloud (S3 RRS in Virginia) and the local NFS; the output is stored locally only. In this case the cost of data transfer exceeds the cost of storage for additional replicas; again the Amazon cluster in the region comes out as the best option for cost/latency. Solving time for this example is 148 ms; the ILP problem contains 94 variables.

B. Scalability of the solver

In this subsection we show the scalability of our approach when the number of variables starts increasing. This aspect is very important since we are dealing with an NP hard problem. More complex examples than in the previous subsection are certainly possible: the number of cloud providers could increase in the future, cloud providers will launch new storage services, new datacenters will be built and applications may include more datasets. The potential increase for each of these factors is also limited, though: the space for potential new cloud providers is limited (it requires capital to build datacenters and the software infrastructure); cloud providers cannot develop and offer support for a large number of storage abstractions; there are constraints in the placement of new, big datacenters (availability of cheap electricity); and users may have a limited ability to manage multiple sets of data instead of consolidating multiple data with similar characteristics into a dataset to be managed as a unit.

We present our results on **Error! Reference source not found.**. For this graph we generate different storage systems with random costs (normally distributed against around averages such as 10 cents per GB per month for storage cost) and feed the problem model to the solver. In this scenario we generate 4 different cloud providers, each one with a number of datacenters within the United States (from 1 to 6) a several matching storage systems (again, from 1 to 6). We consider an application with 3 different datasets. For example, if we choose 3 datacenters and 4 storage systems, the possible number of storage systems for a datasets is: 4 clouds * 3 datacenters/cloud * 4 storage systems/datacenter = 48 possible storage systems. In this case the ILP solver comes up with a solution in 1.08 seconds, on average. In our worst case scenario there are 144 possible storage systems for each dataset and the average time it takes to solve the allocation problem is 37.49 seconds; right now we believe that for each dataset there may be an order of 10 possible storage systems (0.106 seconds solving time) and that, for the reasons mentioned above, an increase of several orders of magnitude is unlikely. And even if this increase were to take place there are still a number of ways to reduce solving time. One of the most obvious ways is to perform better filtering based on storage capabilities matching since it can greatly reduce the size of the problem. Another option is tuning the ILP solver to the characteristics of our objective function

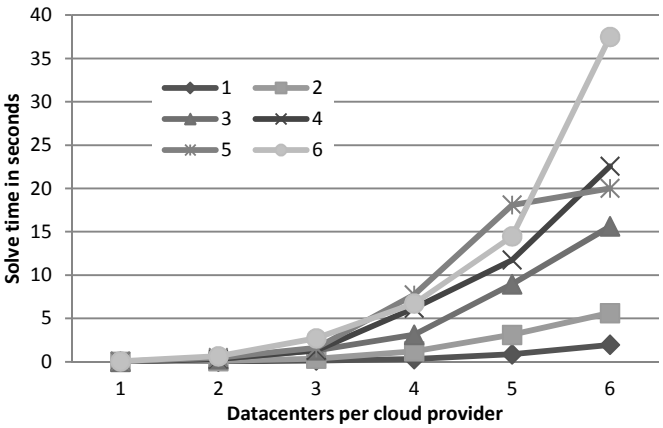

Figure 2. Solve time in seconds as a function of the number of datacenters per cloud provider (X axis) and the number of storage systems available per cloud provider and per datacenter (graph lines). Each data point represents the average over 20 runs.

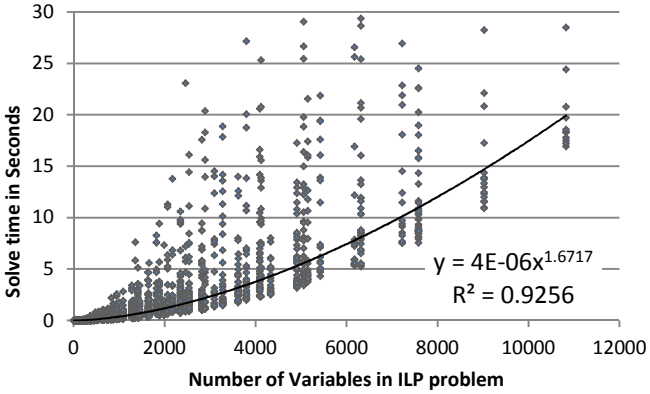

Figure 3. Solve time in seconds as a function of the number of variables in the data allocation problem formulation. One data point represents a single solver execution that returns a 2-tuple (# variables, solve time). A fitted power equation is also included.

and constraints. Right now we use the lp_solve solver with the default settings with one customization: we add a pre-solve stage that deletes variables dominated by other variables. Further customizations may be possible or even using a complete different solver. However, we consider that these results meet our requirements and no further optimizations are needed.

In **Error! Reference source not found.** we chose to set a constant number of cloud providers and application datasets; we can generate an n-dimensional graph in which we vary these two parameters too. However, we think that a more clear representation will be to plot the solving time against the number of variables in the problem model, like in **Error! Reference source not found.**. The parameters number of cloud providers and number of datacenters per cloud provider affect the number of possible storage systems. The size of the ILP problem is based on this number:

$$N.of\ variables = \sum_i datasets_i * (storage\ systems_i * compute\ sites + storage\ systems_i) \quad (15)$$

The data in **Error! Reference source not found.** comes from running different scenarios, which include variations in the number of cloud providers, datacenters and storage

systems. This figure shows a relationship between the number of variables and solving time that can be approximated by a power function, which fits the data very well (R^2 is greater than 0.92). Since the exponent of this power function is small ($x^{1.67}$), problem sizes with a few thousand variables can be solved fast. The same figure also shows that as the problem size increases the variability of the results does too.

In summary, we believe that, given problem sizes based on current cloud offerings, the data allocation problem in cloud computing can be solved in under a second. Future growth of cloud providers and interfaces may push this threshold to half a minute if there is an order of magnitude increase; these results were generated with a standard desktop machine and make no assumptions regarding performance improvements of future ILP solvers or the development of new heuristics.

C. *Sensitivy of the solution*

Previously we have described how we arrive to an optimal solution based on the inputs from the user and the current state of the cloud providers. In this subsection we discuss how this solution is affected by the inputs. We have considered three factors that affect the user's confidence on a given solution: the variability of cloud providers' cost and performance; the user-provided weights in our objective function; and the accuracy of the user's estimation of data requirements. Over the long term the performance and cost of different cloud providers will vary; however we consider this to be a factor that is too difficult (or impossible for the user) to predict and that its short term variability is small. Price changes are infrequent and our experience seems to show that performance over the short term (weeks) is, on average, mostly stable [2].

The user-provided weights for our objective function will have a much greater impact. For example, a user may select the weights for the storage costs, compute costs, latency and bandwidth to be 0.30, 0.30, 0.20, and 0.20, respectively. How does the user select these quantities and not 0.25, 0.30, 0.25 and 0.20? Would that lead our system to arrive to a completely different solution? Essentially we have here a 4-dimensional space in which each point represents the data allocation solution. Since our solver is fast, we can choose to re-run the solver with the different parameters and compare the new solution with the given one; if they are the same we consider these two points to be in the same volume (which represents a data allocation). In order to provide a visual representation of this, we have run our first example in Section 0 A with different weights for storage, compute and latency. We start with data point (0.33, 0.33, 0.33) and process its neighbors; if they are the same solution we add the point to the output and recurse; the 3D convex hull of these points is shown in Figure 4. From this data set we can also find out the limit values for each alpha: all other alphas being equal, what is the range for each alpha that maintains the same solution? These ranges are: for the storage cost [0.26, 0.42], for the compute cost [0.275, 0.375] and for latency [0.29, 0.37]. In this example we do have a medium range of values whose solution is shared; in other problem instances we may have a much smaller (or larger) range. We want to emphasize that a short range is not necessarily a bad option; if the user is confident about the weight values then we give a correct and optimal solution.

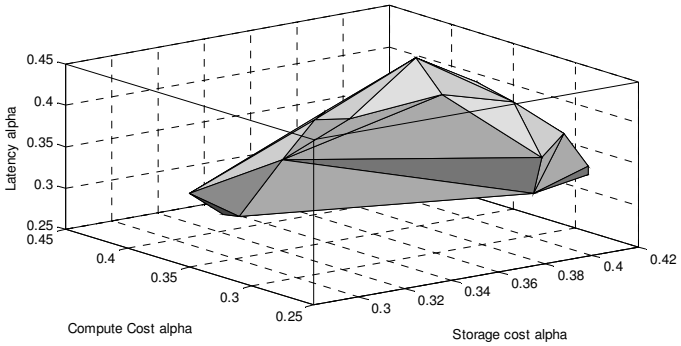

Figure 4. Volume that represents the alpha values for which the same optimal solution exists. The starting values are 0.33 for all storage, compute and latency alphas.

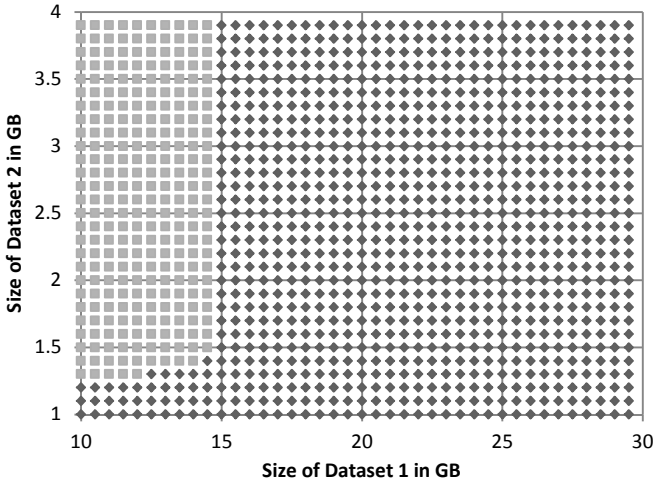

Figure 5. Solution space based on the sizes of datasets 1 and 2 for example in section IV A. Each of the two marker types represents a different data allocation solution.

However, in cases where these weights are a ballpark estimate the graphs and numerical ranges shown should be useful.

We end this section with a discussion of the user-provided data requirements. In many instances it is difficult to estimate the output size of an application, or the number of application runs (and their length). Similarly to our discussion of the weights (alphas) we can re-run our solver varying different input parameters; we continue using the same example. In this case we have chosen the sizes of the first and second datasets. We assign each dataset a range of possible sizes: for dataset one from 10 to 30 GB and for dataset 2 from 1 to 4 GB; each point in this 2D space represents a data allocation solution. We have represented the data in Figure 5. We start with the original solution in the bottom left corner of the graph and compare it to each other solution. If they are the same we use a diamond marker, otherwise we use a different marker. As it turns out in this space there is only one other possibility; this other solution becomes the optimal one as we increase the size of the second datasets while the size of the first dataset remains under 15 GB. This graph compares only storage allocation decisions; compute (job scheduling) is not considered here.

In summary, the solution that our system finds for a concrete data allocation problem is an optimal one, but we recognize that the inputs to this problem may not be exact. In order to avoid a *garbage in, garbage out* type of situation we

re-run the solver with incrementally slight variations of the input parameters and compare the outputs. In this section we have explored a couple of data representations that can provide the user with information on how stable the solution is. This analysis is possible because of the fast solving stage; we can analyze hundreds of data points and generate the graphs shown in a few minutes.

V. USE CASES

In this section we present two possible use cases with two scientific applications, BLAST and MODIS. In the first use case (BLAST) we modify our formula to add a budget constraint. Thus, a use can ask our system to give the best solution given a budget of $250 per month (or any other figure). In the second use case (MODIS) we show how we can modify our formula beyond the cost (storage and compute), latency, and bandwidth terms. Here we add computational length as a term so the user can ask for the solution with the shortest job completion time, given a set budget (and in addition to the usual data requirements).

A. BLAST

The Basic Local Alignment Search Tools is a very popular algorithm in bioinformatics which enables the search of genetic sequences. We use the following parameters for the datasets and the computation requirements: 20 GB input dataset (approximately the size of the publicly available human databases), 30 seconds query time, and 30 KB of output in table format per query. We want to find out, with a limited budget, what is the best solution for a set number of queries per month. In order to do so we will modify the problem formulation by moving the storage and compute costs from the objective function (2) to a new linear constraint:

$$Budget \geq Storage\ Cost + Transfer\ Cost + Request\ Cost + Compute\ Cost \quad (16)$$

The first three costs are defined in equations (4), (5) and (6) and the Compute Cost term is equal to:

$$Compute\ Cost = \sum_k computation_k \times costHour_k \quad (17)$$

We run different scenarios that are represented in TABLE II. Each run of our prototype is given a budget and a number of queries and returns the data allocation for both the input and output datasets. We iteratively increment the number of queries per month (5,000 more each step) till the system is not solvable; each row of the table shows the scenario with the maximum number of queries.

TABLE II. BLAST ON A BUDGET

Monthly Budget	*Input Dataset*	*Output Dataset*	*Number of Queries*
$100	AWS S3 RRS, VA Local NFS, VA	Local SQL, VA	60,000 (local) 15,000 (EC2, VA) 75,000 (total)
$250	AWS S3 RRS, VA Local NFS, VA	Local SQL, VA	60,000 (local) 50,000 (EC2, VA) 110,000 (total)
$500	AWS S3 RRS, VA Local NFS, VA	Local SQL, VA	60,000 (local) 110,000 (EC2, VA) 170,000 (total)
$1000	AWS S3 RRS, VA Local NFS, VA	Local SQL, VA	60,000 (local) 225,000 (EC2, VA)

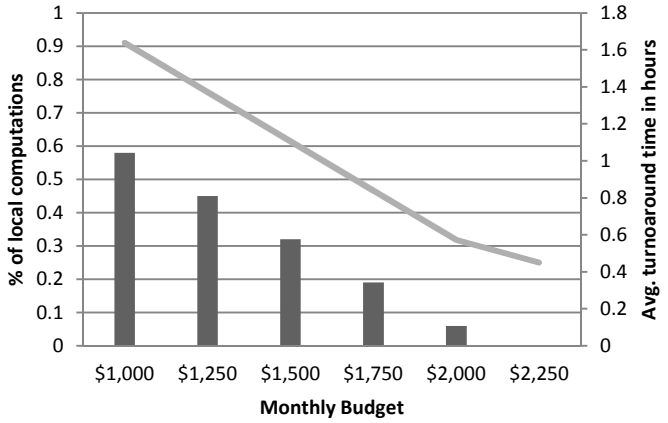

Figure 6. Relationship between the average turnaround time of jobs processing one year's data and the monthly budget. We also show the percentage of local computations: as the budget increases better machines are rented from the Azure cloud and turnaround time improves.

Monthly Budget	*Input Dataset*	*Output Dataset*	*Number of Queries*
			285,000 (total)

All the solutions share the same data allocations: the input dataset is replicated in the local cluster (NFS) and in the Amazon datacenter in Northern Virginia (S3 Reduced Redundancy Storage); the output dataset is stored in a local MySQL database. For query processing the local machine is maxed out in every case at 60,000 queries per month; additional compute power is allocated in Amazon. The different budget levels give us the maximum number of queries; this takes into account the cost of the data replication, the transfer of output data for the computations carried out in Amazon, and the cost of the EC2 machines. In this example we consider a local machine to be equal in power to the Amazon EC2 medium instance; in the next section we show how to take into account the differences between instance types.

B. MODIS Cloud Bursting

Our next example uses the MODIS Azure scientific application. This application processes satellite data from several years to present analysis on certain processes, for example evapotranspiration on the earth surface. A previous paper [17] has presented the performance results of the application running on Windows Azure, the local Windows HPC cluster, and a combination of both. The combination of local and cloud resources is labeled *cloud bursting*; the paper present performance numbers for tasks that are allocated in the cloud and need to find the input data (stored locally in blobs or remotely in files). This paper concludes the evaluation section with "We have found that in general the determining factor is data –where it is and how much is moved. In many situations the key to successful cloud bursting is to minimize data movement". Hence, we believe that this application can benefit from our data allocation algorithms.

In this case we are not considering latency or bandwidth as important metrics; we use average computation length (or turnaround time) instead. Thus, our first step is appending the following term in our general formula (1):

$$w_5 * Average\ Computation\ Length =$$

$$\frac{\alpha_5}{avg.compute\ time} * \frac{\sum_{k,h} compute_{k,h} * speed_{k,h} * compute\ length}{\#\ of\ compute\ hours\ per\ month} \quad (18)$$

We changed the variable $compute_k$ to $compute_{k,h}$; this variable now means "number of monthly computations on $site_k$ using $profile_{k,h}$". We think of a profile as a different running configuration; for example one profile could be 8 extra-large workers on Azure, and another one 32 medium Azure workers. The computation length is the time it takes to complete in the standard local profile; the $speed_{k,h}$ modifiers come from benchmarking. Given this modification, and the one presented in the previous subsection, we can ask our system to give us the best data and compute allocation that give us the fastest turnaround time for jobs for a given budget.

The input parameters are the following ones: each year's data is separated into day files; on average each day has 2.96 GB of input data and its process generates 5.70 MB of output after using 416 MB of temporary storage; we store the data for years 2000 to 2010. Each computation access a complete year, that is, $1/10^{th}$ of each dataset and its length depends on the machine being run on. The actual numbers used for these input parameters are taken from the referenced paper. The solution allocates the input and output data in both the Azure Blob (US North Central datacenter) and the local HPC Cluster. The computation is done by the local cluster nodes and medium size nodes in Azure. Figure 6 presents two measurements regarding the computation (at different budget levels): the percentage of application runs done in the local cluster and the average turnaround time for each application run (which processes and reduces one year of satellite data). The lower the budget the more computations we do locally and the slower these computations are. Given this information about this tradeoff (cost vs. speed) the user can make sound allocation decisions based on her preferences or requirements.

VI. Limitations and Future Work

As we have seen in the previous sections, our approach relies on having accurate information on the capabilities of the cloud providers. Currently there are multiple websites that continuously benchmark cloud providers like Amazon or Azure; we believe that the community will greatly benefit of having a more thorough approach with more metrics and making the data machine consumable (as opposed to web graphs). Another limitation that is present for developed applications is that interfaces are different across clouds; it is difficult to modify an application to make it possible to run on different clouds. The solution to this issue will probably come by having cloud-agnostic APIs for data access (such as CSAL) and by introducing more compatibility at the execution level: running arbitrary apps for Platform as a Service providers (Windows binaries on Azure), having compatible APIs (Eucalyptus and Amazon) or other ports (Google App Engine on Amazon EC2).

One final limitation is related to our allocation of computation. In this paper we have introduced a planning phase that gives us a data allocation solution and a coarse-grained approach to computation: we do not take into account factors such as the hourly billing of cloud providers, the VM startup time and the shape of the computation (single-threaded, workflow, etc.). We believe that all these factors are better accounted for with an online approach. Thus, the next step in our work is the *data-aware Scheduler* component (Figure 1). We would like to explore dynamic algorithms that make the actual scheduling decisions as the job request come. In addition to this scheduling phase we would like to explore the possibility of integrating our resource allocation solutions with a distributed data system (for example, iRODS) so we can automate our approach further.

VII. Conclusion

In this paper we have presented our approach to data allocation in cloud computing. Building upon our previous work, where we match each application dataset with a set of possible storage services based on storage capabilities and data requirements, we first generate an integer linear programming problem that takes into account storage and compute costs, latency, and bandwidth. This ILP problem model takes into account the unique characteristics of cloud computing. Our software implementation uses an ILP solver to find an optimal data allocation solution in one second or less; we have also shown that our approach is scalable as the number of cloud providers, datacenters or storage services increase. These short running times also allow us to gather more information about the sensitivity of this solution and present it to the user. Finally we have presented two use cases with the BLAST and MODIS applications. Small changes in our problem formulation allows us to add a monthly budget restriction and to minimize job turnaround time in our optimal solutions; combining local and cloud resources we can halve the cost compared to a cloud-only approach or increase job turnaround time by 52% compared to a local-only approach..

VIII. References

[1] M. Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing." 2009.

[2] A. Ruiz-Alvarez and M. Humphrey, "An Automated Approach to Cloud Storage Service Selection," in *2nd Workshop on Scientific Cloud Computing (ScienceCloud 2011)*, 2011.

[3] M. Berkelaar, K. Eikland, and P. Notebaert, "lpsolve: Open source (mixed-integer) linear programming system," 2011. [Online]. Available: http://lpsolve.sourceforge.net/.

[4] W. W. Chu, "Optimal File Allocation in a Multiple Computer System," *IEEE Transactions on Computers*, vol. 18, no. 10, pp. 885-889, Oct. 1969.

[5] R. G. Casey, "Allocation of copies of a file in an information network," in *In Proceedings of the AFIPS Joint Computer Conferences*, 1972, pp. 617-625.

[6] E. Grapa and G. G. Belford, "Some theorems to aid in solving the file allocation problem," *Communications of the ACM*, vol. 20, no. 11, p. 878, 1977.

[7] K. Lam and C. T. Yu, "An approximation algorithm for a file-allocation problem in a hierarchical distributed system," in *In Proceedings of the 1980 ACM SIGMOD International Conference on Management of Data*, 1980, pp. 125 - 132.

[8] S. Mahmoud and J. S. Riordon, "Optimal allocation of resources in distributed information networks," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 1, p. 66, 1976.

[9] L. W. Dowdy and D. V. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys (CSUR)*, vol. 14, no. 2, p. 287, 1982.

[10] B. Gavish and O. R. L. Sheng, "Dynamic file migration in distributed computer systems," *Communications of the ACM*, vol. 33, no. 2, p. 177, 1990.

[11] B. Awerbuch, Y. Bartal, and A. Fiat, “Competitive distributed file allocation,” in *Annual ACM Symposium on Theory of Computing*, 1993, pp. 164-173.

[12] H. L. Morgan and K. D. Levin, “Optimal program and data locations in computer networks,” *Communications of the ACM*, vol. 20, no. 5, p. 315, 1977.

[13] Lintao Zhang and Sharad Malik, “The Quest for Efficient Boolean Satisfiability Solvers ,” *Computer Aided Verification*, vol. 2404, pp. 641-653, Sep. 2002.

[14] Microsoft, “Microsoft Solver Foundation.” [Online]. Available: http://msdn.microsoft.com/en-us/devlabs/hh145003.aspx.

[15] Z. Hill, M. Mao, J. Li, A. Ruiz-Alvarez, and M. Humphrey, “Early Observations on the Performance of Windows Azure.,” in *1st Workshop on Scientific Cloud Computing (ScienceCloud 2010)*, 2010.

[16] A. Ruiz-Alvarez and M. Humphrey, “ILP problem formulation and solutions,” 2011. [Online]. Available: http://www.cs.virginia.edu/~ar5je/ILP.html.

[17] M. Humphrey, Z. Hill, K. Jackson, C. van Ingen, and Y. Ryu, “Assessing the Value of Cloudbursting: A Case Study of Satellite Image Processing on Windows Azure,” in *7th IEEE International Conference on e-Science (escience 2011)*, 2011.