

# Reshaping Text Data for Efficient Processing on Amazon EC2

Gabriela Turcu  
Computer Science  
Department  
University of Chicago  
Chicago, Illinois 60637  
gabri@cs.uchicago.edu

Ian Foster  
Argonne National Laboratory  
Computer Science  
Department  
University of Chicago  
Chicago, Illinois 60637  
foster@anl.gov

Svetlozar Nestorov  
Computation Institute  
Chicago, Illinois 60637  
evtimov@ci.uchicago.edu

## ABSTRACT

Text analysis tools are nowadays required to process increasingly large corpora which are often organized as small files (abstracts, news articles, etc). Cloud computing offers a convenient, on-demand, pay-as-you-go computing environment for solving such problems. We investigate provisioning on the Amazon EC2 cloud from the user perspective, attempting to provide a scheduling strategy that is both timely and cost effective. We rely on the empirical performance of the application of interest on smaller subsets of data, to construct an execution plan. A first goal of our performance measurements is to determine an optimal file size for our application to consume. Using the subset-sum first fit heuristic we reshape the input data by merging files in order to match as closely as possible the desired file size. This also speeds up the task of retrieving the results of our application, by having the output be less segmented. Using predictions of the performance of our application based on measurements on small data sets, we devise an execution plan that meets a user specified deadline while minimizing cost.

## Categories and Subject Descriptors

C.2.4 [Distributed Computing]: Cloud Computing—*Provisioning*

## General Terms

Performance, Design

## Keywords

Cloud Computing, Provisioning, Amazon EC2, Text Processing

## 1. INTRODUCTION

As the amount of available text information increases rapidly (online news articles, reviews, abstracts, etc.), text analysis applications need to process larger corpora. Increased

computational resources are needed to support this analysis. Building and maintaining a cluster requires significant initial investments (hardware, physical space) and operational costs (power, cooling, management). Amortizing these costs demands for high utilization of the resources. This however limits the ability of projects to grow their resource needs when necessary. Recently, commercially offered cloud computing [8] solutions (Amazon EC2, GoGrid, SimetriQ, Rackspace) have become an attractive alternative to in-house clusters. They offer many advantages: customizable virtual machines, on-demand provisioning, usage based costs, fault tolerance. Some of the drawbacks are on the side of performance guarantees and security. In a cluster environment the user typically delegates the task of resource allocation to the local resource manager, while the cloud user can take control of this step. We see this as an opportunity to steer application execution in such a way as to meet a user deadline while also minimizing costs.

A considerable amount of recent work has focused on analyzing the performance and cost effectiveness of such platforms for different classes of applications: CPU intensive or I/O intensive scientific computing applications [10, 5, 17, 11], service-oriented applications [6], latency-sensitive applications [3]. Other work has focused on quantifying the variation in received quality of service [12]. Some of this work relies on simulations of a cloud environment, while most of it uses Amazon's Elastic Computing Cloud (EC2) as a testbed.

In this paper, we consider typical text processing applications (`grep`, `part of speech tagging`, `named entity recognition`) and attempt to provide a good execution plan for them on Amazon EC2. Our input data sets consist of a large number of small files. We assume knowledge of the distribution of the file sizes in the input data set, and no knowledge of the internals of the application we are running. Our first goal is to quantify the performance gap suffered by our applications if consuming small files. To achieve this goal we observe the application's behavior on Amazon EC2 for different file sizes and identify a suitable file size or range of sizes. We then reshape our input data by grouping and concatenating files to match the preferred size as closely as possible. The text processing applications we consider do not need to be further modified to be capable to consume the concatenated larger input files. This approach will also imply a lower number of output files which implies a shorter

retrieval time for the application results. This results in a shortened makespan for the application. In terms of cost, the per-byte cost being constant, the only benefit results from the shorter makespan.

A second goal of our work is to use our application as a benchmark on Amazon EC2 to determine a good execution plan for the entire input data. In order to devise a schedule we need estimates of the application runtime on Amazon resources. We observe the application’s behavior on EC2 instances for small subsets of our data and then attempt to determine a predictor of runtimes for larger subsets of our final workload. We consider linear, power law and exponential functions as predictors.

## 1.1 Background

The Elastic Computing Cloud (EC2) from Amazon offers its customers on-demand resizable computing capacity in the cloud with a pay-as-you-go pricing scheme. Amazon relies on Xen virtualization to offer its customers virtual hosts with different configurations. The user can request different instance types (small, medium, large) with different CPU, memory and I/O performance. The instance classification is based on the notion of an EC2 compute unit which is equivalent to a 1.0-1.2 GHz 2007 Opteron 2007 Xeon processor. The user can choose among a range of Amazon Machine Images (AMIs) with different configurations (32-bit/64-bit architecture, Fedora/Windows/Ubuntu). Users can modify AMIs to suit their needs and reuse and share these images.

Amazon allows the user to place an instance in one of the 3 completely independent EC2 regions (US-east, US-west, EU-west). This would allow the user to pick a location closer to where their data is available. Within a region, the users can choose to place their instances in different availability zones which are constructed by Amazon to be insulated from one another’s failure. For example, the US-east region has 4 availability zones (us-east-1a, us-east-1b, us-east-1c and us-east-1d). These zones are defined separately for each user. Amazon’s SLA commitment is 99.95% availability for each Amazon EC2 Region for every user.

Amazon instances come with ephemeral storage (160GB for small instances). Amazon also offers the possibility to purchase Elastic Block Store (EBS) persistent storage. EBS volumes are exposed as raw block devices and can be *attached* to an instance and persist beyond the life of that instance. Multiple EBS volumes may be attached to the same instance, but an EBS volume may not be attached to multiple instances at the same time. The root partition of an instance may be of type *instance-store* in which case its contents are lost in case of a crash, or of type *ebs* in which case its contents are persistent.

Amazon offers storage independent of EC2 via the Simple Storage Service (S3). Users can store an unlimited number of objects each of size of up to 5GB. Multiple instances can access this storage in parallel with low latency, which is however higher and more variable than that for EBS.

The pricing for these services are summarized in the Table 1. We note the pricing scheme for instances where we pay a flat rate for an hour or partial hour ( $\$0.1 * [h]$ ). This

**Table 1: AWS services pricing**

Resource	Type	Pricing
<b>EC2</b> m1.small	Compute	\$ 0.10/hr (or part. hr)
	Transfer in	free (\$0.1/GB after June 2010)
	Transfer out	\$0.15/GB
	Transf. within zone or S3	free
<b>S3</b>	Storage	\$0.15/GB/month
	Transfer in	free (\$0.1/GB after June 2010)
	Transfer out	\$0.15/GB
	PUT	\$0.01 per 1,000 requests
	GET	\$0.01 per 10,000 requests
<b>EBS</b>	Storage	\$0.1/GB/month
	I/O	\$0.1/million I/O requests
	Transfer in	free (\$0.1/GB after June 2010)
	Transfer out	\$0.15/GB

has implications for devising a good execution plan for an application. Once an instance is up and running, we should always plan to let it continue to run at least to the full hour unless this prevents us from meeting the user deadline.

Amazon has also started to offer *spot instances* as of December 2009. The price for these instances depends on current supply/demand conditions in the Amazon cloud. The user can specify a maximum amount he is willing to pay for a wall-clock hour of computation and can configure her instance to resume whenever this maximum bid becomes higher than the current market offer. This is advantageous when time is less important of a consideration than cost. Applications are required to be able to resume cleanly in order to best take advantage of spot instances. In our work, we are interested in being able to give cost effective execution plans when there are makespan constraints and so we use instances that can be acquired on demand.

## 2. MOTIVATION

Our work is motivated by the computational needs of a project analyzing a large collection of online news articles. While the size of a single article is relatively small (a few dozen kilobytes), the total number of articles (tens of millions) and total volume of text (close to a terabyte) make the efficient processing this data set challenging. In particular, we consider the idea of reshaping the original data, characterized by millions of small fragments with significant size differences, into large blocks of similar size. Processing these large blocks in parallel in the cloud is more attractive than dealing with the original data for two reasons. First, much of the overhead of starting many new instances and processes is avoided, making the overall processing more efficient. Second, the execution times for the similarly-sized blocks of data may also be relatively similar, thus enabling the estimation of the total running time and the optimization of the cost for typical pricing schemes given a deadline.

There are many other large collections of text that share the same characteristics as our target dataset. For example, social scientists are interested in the ever-growing myriad of short texts generated by social network activities such as status updates, tweets, comments, and reviews. Bioinformatics researchers often analyze a large number of abstracts,

posters, slides, and full papers in order to extract new and emerging patterns of interactions among proteins, genes, and diseases.

### 3. EXPERIMENTAL SETUP

In this section, we describe the resources we use on EC2 and the characteristics of our data sets.

#### 3.1 EC2 setup

Small instances have been shown to be less stable [6, 18, 3] but more cost effective. Our experiments use small instances since they are most common and most cost effective. We use a basic Amazon EC2 32-bit small instance running Fedora Core 8. Each such instance is configured with 1.7 GB memory, 1 EC2 compute unit, 160GB local storage, 15GB EBS root partition. The cost of an instance is \$0.1 per hour or partial hour. Payment is due only for the time when the instance is in the *running* state and not while it is starting up (*pending* state) or shutting down (*shutting down* state) or once it is in the *terminated* state.

We use the local instance storage for most of our experiments. Using EBS volumes, though adding to the cost of execution, has an advantage in simplifying how the execution plan would adapt to failure or bad performance. If we decide an instance is not performing well, we may decide to let it run to the full hour while starting up another instance and attaching the EBS volume to it once it is ready. For an I/O intensive application, a simple calculation shows that if working with a slow instance with an average read speed of 60 MB/s, we could process approximately 210GB of data if we let the instance run for the next hour. If switching to another instance that is likely fast and consistent, even when paying a penalty of 3 minutes for the new instance startup and EBS volume attachment (which we might minimize by starting this instance while the previous one is stopping), we would still be able to process an extra 57 GB. If the instance happens to be slow we miss processing 10 GB.

#### 3.2 Data

We use two data sets in our experiments. The first is a set of HTML articles that are part of our Newslab collection. The Newslab data comprises of roughly 75 million news articles collected from Google News during the year of 2008. We use a subset of this data that corresponds to English language articles. This set comprises of approximately 18 million files adding up to a volume of almost 900GB. The majority of the files are less than 50KB and the distribution of the file sizes exhibits a long tail. The largest file size is 43MB. Figure 1 shows the distribution up to files of size 300KB. The file sizes are considered as multiples of 10K.

The second data set consists of 400000 English language text files, extracted from a subset of HTML English language articles. The majority of the files are small (<5KB), while the largest file is 705KB in size. The plot below shows the frequency distribution of the sizes of the files up to 160KB. The distribution has a long tail (Figure 2).

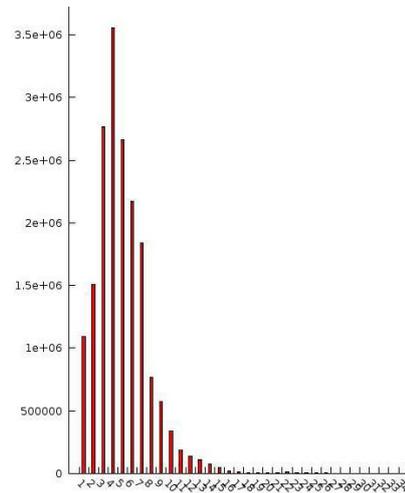


Figure 1: Frequency distribution for the HTML dataset HTML\_18mil (10KB bin)

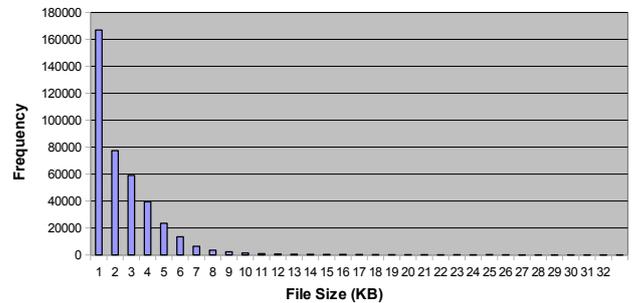


Figure 2: Frequency distribution for the text dataset Text\_400K (1KB bins)

### 4. PERFORMANCE ESTIMATION

Any execution strategy for an application on a set of resources relies on the expectation of how the application performs on each resources. Performance estimation can be done through analytical modeling [13], [4], empirically [7] and by relying on historical data [15]. In our setup, we have knowledge of the characteristics of the data set, but no knowledge of application behavior.

Our approach is to first request a small instance and measure its performance using *bonnie++* [1] to ensure that it is of high quality (over 60MB/s block read/write performance). We repeat the performance measurement to check if the behavior of the instance is stable in time. This turns out to be the case for most instances. We repeat this step until we receive an instance that performs well.

We then send small probes of our data set to the local storage of the instance. Initially we send a single file (probe  $P_{orig}^{V_0}$  of volume  $V_0$ , in its original form) and measure the execution time the application on that input. We pick the initial file to send to be among the smallest in our data set. We repeat the application performance measurements 5 times and keep the average and standard deviation. If the average value is small and the standard deviation is large, we continue to profile

the application performance for larger volumes of data.

The next step is to carve out a larger volume  $V_1 = k * V_0$ , with an appropriate  $k$  based on the amount of time taken to process the initial probe. From the original probe for volume  $V_1$ ,  $P_{orig}^{V_1}$  we use the first fit bin packing algorithm to merge the original files into desired unit file sizes  $(s_0, \dots, s_n)$ . We pick  $s_0$  larger than the maximum file size in the original set. We then conveniently choose  $s_1 \dots s_n$  as multiples of  $s_0$ , such that we perform the bin packing once to obtain  $P_{s_0}^{V_1}$  and then directly derive the remaining probes  $P_{s_1}^{V_1} \dots, P_{s_n}^{V_1}$ . This is more convenient since we avoid rerunning the first fit bin packing algorithm, but is sensitive to the quality of the original bins of size  $s_0$ . We vary the base file size up to the maximum possible size of  $s_n = V_1$ . We then analyze the performance of the original probe  $P_{orig}^{V_1}$  and contrast it with the results for the other probes in order to learn of any performance loss or gain that we would incur if the same data was organized in smaller or larger files.

If the results for the set of probes  $(P_{orig}^{V_1}, P_{s_0}^{V_1}, P_{s_1}^{V_1}, \dots, P_{s_n}^{V_1})$  are not yet stable we continue this process with larger volumes. At the end of this process, we obtain measurements along three dimensions: data volume (corresponding to each probe set), file unit size (corresponding to each element in a probe set) and execution time.

Collecting the results for all the sets of probes we have, we can inspect each probe set to identify a possible preferable file size where the execution time is minimal. Sometimes we do not observe a single global minimum for a curve, but rather a plateau where the execution time is minimized. We give preference to choosing the preferred file size unit as the minimum from later probe sets that are more stable.

## 5. STATIC PROVISIONING

The earlier experiments allow us to determine a best file size unit or a range of file size units that perform better than the original. Once we have selected a preferred file unit size, we consider the data points relevant to that file size unit from each probe set. We use these data points to perform regression to obtain a predictor for execution times as a function of data volume consumed. While this is a simple approach, we believe we can get a satisfactory estimate of the runtime without investing in determining complex performance models. Since our data points are not nearly equidistant, we perform the regression in logarithmic space. We attempt to fit the following functions:

1. *Linear*  $y = ax$  : In logarithmic space, we would be fitting  $Y = \ln a + X$ , where  $Y = \ln y$ ;  $X = \ln x$
2. *Power law*  $y = ax^b$  : In logarithmic space:  $Y = \ln a + bX$ . We also fit functions of the form  $Y = aX^2 + bX$  which correspond to original functions  $y = x^{a \ln x + b}$
3. *Exponential*  $y = ae^{bx}$  : In logarithmic space:  $Y = \ln a + bx$

If we obtain a good fit through these means, we can use the predictor to estimate the total execution time of the application  $T$  for the entire volume of data that needs to be

processed  $V$ . We assume the instances are uniform, though this is not the case in reality. We plan to extend our models to account for variability of the instance performance in future work.

We also assume that the data is already staged onto EBS volumes for the grep application and can be staged onto the local storage of the instances for the POS tagging application in a constant time per run, assuming that the bottleneck is the maximum throughput available at the upload site. The pricing scheme considers a flat rate  $r$  (0.085\$ for small instances) for a full or partial hour of computation.

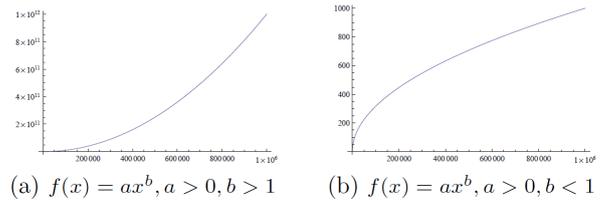
Then, for a given deadline  $D$ , and a linear fit  $y = ax$ :

- If  $D \geq 1$ , then the cost is  $\lceil P \rceil \times r$ . If we ignore boot up time cost of the instances, then this would be equivalent to giving an hour's worth of computation for each instance and a partial hour to the last instance. This would also be the case if we pack  $\lfloor D \rfloor$  hours of computation into each instance (since the constant slope "a" ensures we process the same volume of data in either case)
- If  $D < 1$ ,  $D >$  time taken to process largest (unsplitable) file, then the cost is  $\lceil \frac{P}{d} \rceil \times r$ , where we have no choice but to pay a full hour for instances running for time  $D$ .

$$f(d) = \begin{cases} r \lceil P \rceil & : d \geq 1 \\ r \lceil \frac{P}{d} \rceil & : d < 1 \end{cases}$$

Further, we may repeat this process on non-overlapping subsets of the total volume of data. This would allow us to explore a larger volume of our data set through random sampling, at a smaller computational cost.

In general, we can improve our execution plan by considering more closely the performance models we derived. The figures below show possible shapes for the fitted curves.



**Figure 3: Execution time as a function of data volume**

For  $a > 0, b > 1$  ( $f'' > 0$ ) (Figure 3a), if startup time is small enough, it will always be better to start a new instance, since in a one-hour time slot we can process more data at smaller volumes than at larger volumes.

For  $a > 0, b < 1$  ( $f'' < 0$ ) (Figure 3b), it will always be better to pack as much data as possible by  $\lfloor D \rfloor$  than start a new instance. We will have to compare the volume of data that can be processed between times  $\lfloor D \rfloor$  and  $D$  to the

volume that can be processed in 1 hour from time 0 to 1 to decide which option is cheaper.

## 5.1 Grep

We run `grep` (GNU `grep` 2.5.1) on our first dataset consisting of HTML files from the NewsLab data. `Grep` searches the files provided as input for a matches of a provided pattern. The CPU - I/O mix of `grep` is heavily influenced by the complexity of the regular expression we are searching with and the number of matches found. Complex search patterns can tip the execution profile towards intense memory and CPU usage. Another factor is the size of the generated output which depends on the likelihood of finding a match and the size of the matched results.

We restrict ourselves to the usage scenario of searching for simple patterns consisting of English dictionary words. In our experiments we search for a nonsense word to increase as much as possible the likelihood that it is not found in the text. For a word that is not found we are sure to traverse all the data set regardless of other settings for `grep`, while also isolating from the cost incurred when also generating large outputs.

We set our initial probe  $P_0$  to a volume of 1MB. Figure 4 shows the average execution times. We notice that the values are very small and the standard deviation over 5 measurements is large. We discard these results as too unstable and increase the volume of the probe.

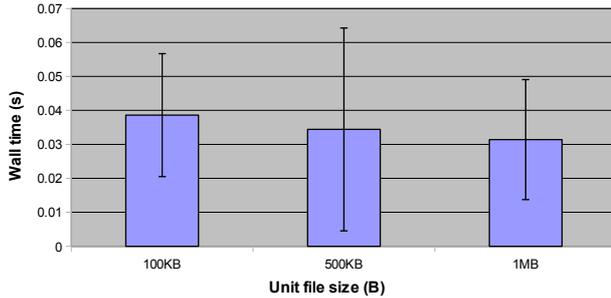


Figure 4: Execution times for `grep` on a 1MB volume

We gradually increase the volume of the probe and observe that the downwards trend continues for larger volumes and file size units. We notice that at the file size unit of 10MB we generally reach a plateau up to 2GB (Figure 5).

A more careful sampling of the file size unit range reveals that the plateau is not smooth as shown in Figure 6. We observed spikes where the performance was degraded. The results are repeatable and stable in time, which rules out a contention state for the networked storage. Our hypothesis is that our probes, while on the same EBS logical volume, were placed in different locations some of which have a consistently higher access time. We verified that this is indeed a possible cause by consistently observing that creating a clone of a large sized directory can result in performance variations of up to a factor of 3.

We select the file size unit to be 100MB which is in the

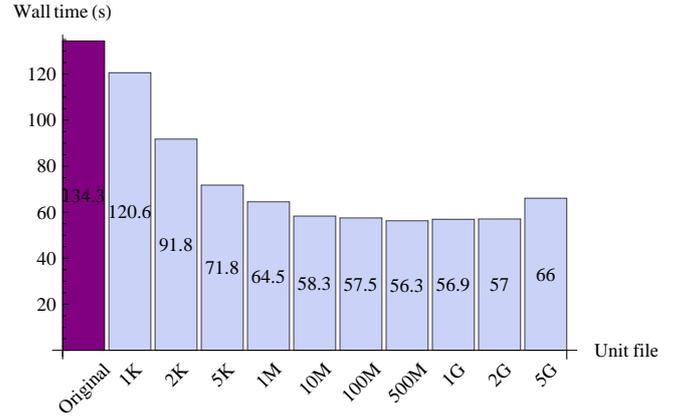


Figure 5: Execution times for `grep` on a 5GB volume

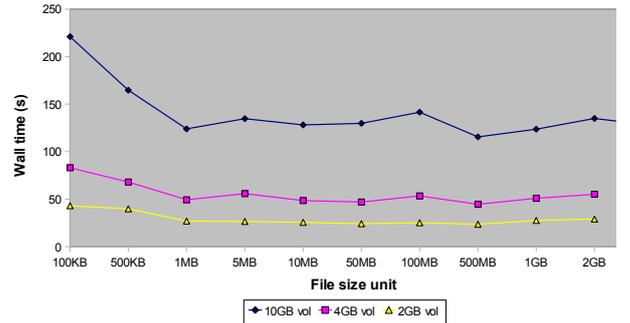


Figure 6: Execution times for `grep` on 1GB, 2GB and 10GB volumes

minimum range and for which our experiments also have a small standard deviation. Based on the measurements we have already collected for the split level of 100MB, we obtain a very good linear fit ( $R^2 = 0.999$  and very small residuals of magnitude  $< 1$ ).

$$f(x) = -0.974 + 1.324 * 10^{-8}x \quad (1)$$

We perform our experiments on a random 100GB volume of the dataset **HTML-18mil** and stage in this data equally across 100 EBS volumes. The deadline we wish to meet dictates how to attach the available volumes to the required number of instances. The unit of splitting of the data across the EBS volumes determines the coarseness of deadlines we can meet.

Let  $V$  be the total volume of 100GB,  $V^0 = \frac{V}{100}$  be the volume on each EBS device and  $f^{-1}(D) = V_D$ , the volume predicted by our model that is required to meet a deadline  $D$ . If we consider a deadline  $D < 1$ , if  $V^0 > V_D$ , we can not directly meet this deadline without reorganizing our data to lower the unit volume  $V^0$ . If  $V^0 < V_D$ , we can provide  $\lfloor \frac{V_D}{V^0} \rfloor$  EBS devices each of volume  $V^0$  to an instance. This would demand that we use  $\lceil \frac{V_D}{V^0} \rceil = i$  instances. We can further improve the likelihood of meeting the deadline by balancing

the volume across the  $i$  instances or by lowering the deadline to be met and reevaluating the execution plan as described in the next section .

Based on our model given by equation (1), we predict that processing  $100GB$  of data within  $D = 3600$  seconds only requires  $1387.8$  seconds. The actual execution time is  $1975.6$ . Figure 7 shows that we underestimate the deadline by almost 30%. The figure also shows a 5.6 fold improvement on the execution time when working with  $100MB$  files instead of the files in their original format of a few kilobytes in size.

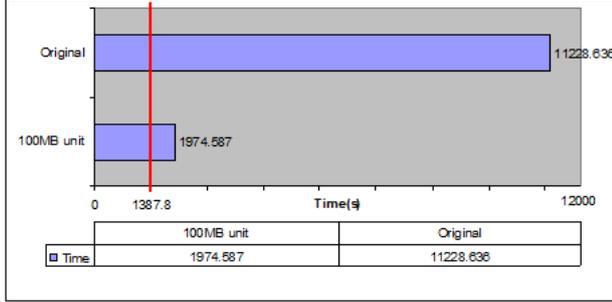


Figure 7: Execution times for grep for 100GB

A possible source of improvement for the predictive power of our performance model, is to consider random samples from our entire dataset and reestimate our predictor. From our data set, we choose 10 random samples (without replacement) of  $2GB$  and measure the execution time of grep on these samples, and a few of their smaller subsets. We consider these samples already in the chosen  $100MB$  file unit size. The measurements show considerable variability: for the 10 samples, at the  $2GB$  volume, we obtain a minimum processing time of  $23.25$  seconds, a maximum of  $45.95$  seconds, average of  $32.2$ . We further refit our model to the new observations and obtain:

$$f(x) = 0.208 + 1.503 * 10^{-8}x \quad (2)$$

The slightly higher slope of equation (2) improves the predicted execution time to  $1576.44$ , but this only reduces the error from 30% to 20% of the actual execution time.

## 5.2 Stanford Part of Speech tagging

The second application we consider is the Stanford Part-of-Speech tagger [16] which is a commonly used in computational linguistics as one of the first steps in text analysis. It is a Java application that parses a document into sentences and further relies on language models and context to assign a part of speech label to each word.

Our goal is to run the Stanford Part-of-Speech tagger with the `left3words` model on our second data set of  $1GB$  size. We wrap the default POS tagger class that is set up to parse a single document, such that we process a set of files avoiding the startup cost of a new JVM for every file.

We note that over 40% of our files are less than  $1KB$  in size. Based on this, we pick the initial file size unit  $s_0$  to be  $1K$ ,

and let  $V_1 = 1000K$ . Using the subset-sum first fit heuristic, we construct probe sets of volume  $1000K$ . The original probe contains over twice the number of files (2183) as the probe with file size unit of  $1K$  ( $1000$ ). The average execution time over 5 measurements for the probe set is shown in Figure 8.

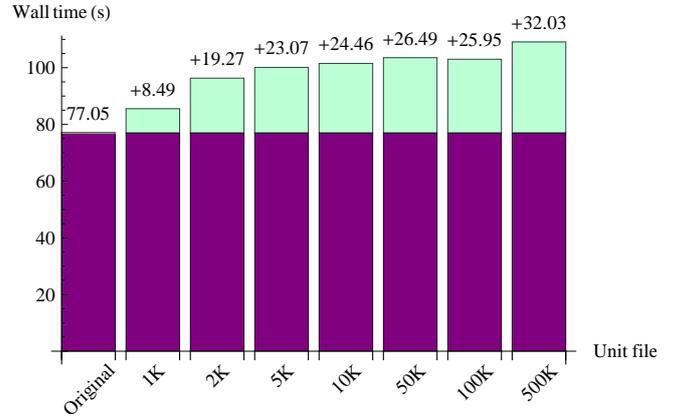


Figure 8: Execution times for POS tagging on a volume of  $1000K$

We observe that the original level of segmentation fairs the best and using a smaller number of larger files does not provide any benefits. The application is memory bound and does not benefit from dealing with larger file sizes.

Keeping the original level of segmentation for the files, we attempt a linear fit of form  $f(x) = ax + b$  for our measurements. We obtain a good fit on which we base our predictions:

$$f(x) = 0.327 + 0.865 * 10^{-4}x \quad (3)$$

Let the total volume of our data set be  $V$ , and the desired deadline be called  $D$ . Using the performance model in equation (3), we attempt to provide execution plans for different deadlines.

For a deadline of one hour ( $D = 3600$ ), we solve equation (3) for  $y = 3600$  and obtain the solution  $x_0$  which represents the amount of data that can be processed within the deadline according to our performance model. The solution prescribes we need  $i_0 = \lceil \frac{V}{x_0} \rceil = \lceil 26.1 \rceil = 27$  instances. We then proceed to pack our data set into 27 bins. For this step, we consider the input files in their original order. If we apply the first fit algorithm to the file sizes sorted in descending order, we are more likely to obtain bins that closely match the prescribed capacity. However, this will result in the first bins containing a small number of large files and the latter bins containing many small files. Our experiments for the POS application show that the degradation for working with large files is pronounced. We therefore choose to consider the files in the order in which they are provided, though improvements are possible considering more refined information about the distribution of the file sizes. With this approach we obtain the result shown in Figure 9:

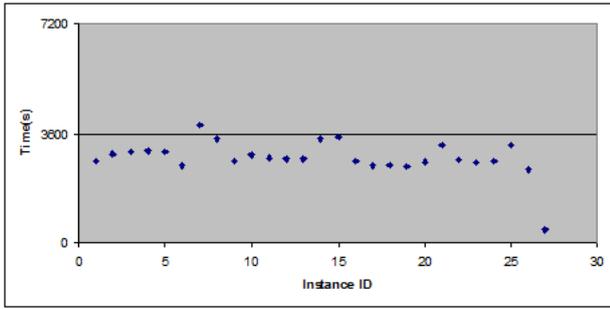


Figure 9: POS tagging for D=1 hour, model (3)

We can improve our schedule, by uniformly distributing the data to each instance (Figure 10). In this way, we reduce the chance of missing the deadline, while still paying the same cost of  $r * i_0$ . With the new bins of size  $\frac{V}{i_0}$  we meet the deadline successfully:

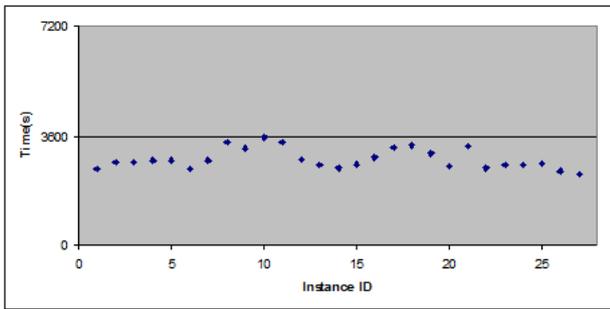


Figure 10: POS tagging for D=1 hour, uniform bins, model (3)

For deadlines larger than 1 hour, if we consider performance prediction models that are linear, exponential or power law and that the instance start up time is insignificant, then the best strategy is to fit an hour of computation into as many instances as needed to complete the task. In reality, the instance startup times are not always insignificant and there are limitations on the number of instances that can be requested. For this reason, we want to find a schedule that also limits the number of instances requested.

When solving equation (3) for  $D = 7200$  and distributing uniformly the data for each instance, we obtain the results in Figure 11, which meets the deadline loosely.

A further improvement for our prediction can be obtained by taking random samples from our data set and reevaluating our performance model. To achieve this, we take 3 samples of 5MB each (without replacement) and measure the execution times for these samples and subsets of them. With the new data points, we obtain another linear fit of good quality:

$$y = 3.086 + 0.725482 * 10^{-4} * x \quad (4)$$

The slope of the new model is lower than that of the model in

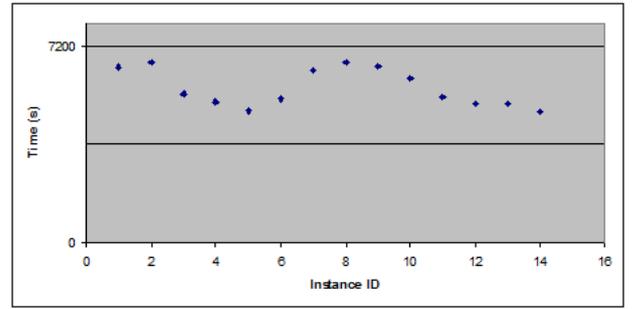


Figure 11: POS tagging scheduling for D=2 hours, uniform bins, model (3)

equation (3), indicating that for the same deadline, the new model will predict we can process more data. This matches the observation that based on the simple linear model from equation (3), we meet the deadline loosely enough that it may be possible that the deadline can be met with a lower number of instances.

Based on the new model in equation (4), we determine we require 22 instances for  $D = 3600$  (compared to the 27 determined by the earlier model) and 11 instances for  $D = 7200$  (compared to the 14 instances required by the earlier model). The results are shown in figures 12 and 13 respectively:

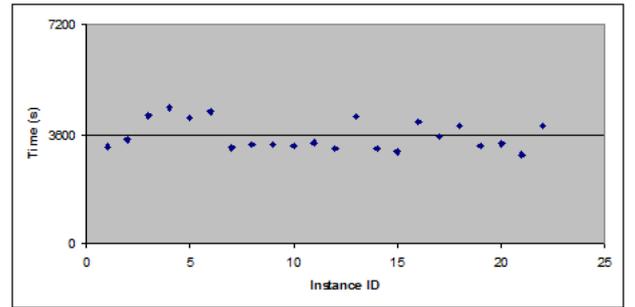


Figure 12: POS tagging for D=1 hour, random sampling, model (4)

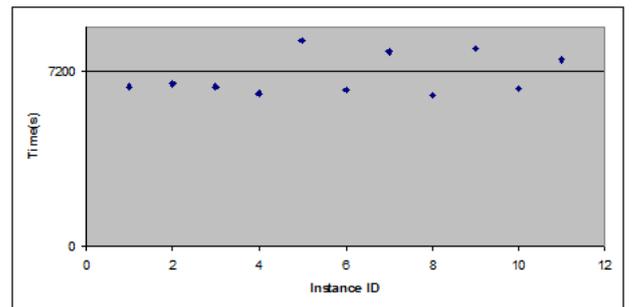


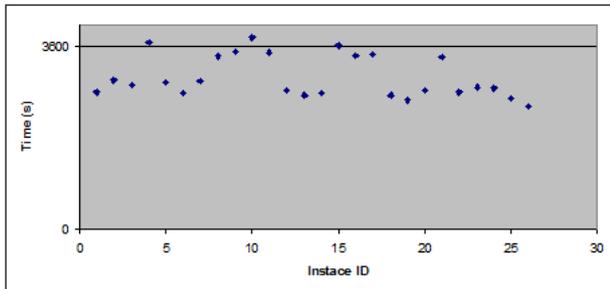
Figure 13: POS tagging for D=2 hours, random sampling, model (4)

We note that the missed deadlines compensate for the benefit would have gotten by using a smaller number of instances.

A reason for missing both deadlines when using the new model (in equation (4)) was that we obtained very full bins, with little opportunity to distribute the data evenly across instances to a lesser volume (and correspondingly lesser deadline) than the one prescribed by  $D$ . When fitting with the earlier model (in equation (3)) we happened to obtain the last bin relatively empty which permitted distributing the data uniformly over the instances at a smaller volume which then corresponds to a lower deadline than that which we must meet.

Based on the residuals for the model in (4), we consider it is acceptable to assume that the relative residuals  $\frac{y-f(x)}{f(x)}$  are normally distributed. We would like to have a small probability of the residual at the predicted value exceeding some quantity. This can be translated in the value  $y$  exceeding a deadline. Assume we would like to have a less than 10% chance to exceed a deadline:  $P(y > D) \leq 0.1$ . Or, in terms of the relative residual:  $P(\frac{y-f(x)}{f(x)} > \frac{D-f(x)}{f(x)}) \leq 0.1$ . Since the relative residual is assumed to be a normal random variable (call it  $X$ ),  $P(X > \frac{D-f(x)}{f(x)}) \leq 0.1$  can be standardized relying on the sample mean and sample standard deviation calculated from the residuals of our model  $\mu_X$  and  $\sigma_X$ . Then,  $P(Z > \frac{\frac{D-f(x)}{f(x)} - \mu_X}{\sigma_X}) \leq 0.1$ , where if  $P(Z > z) \leq 0.1$ , gives  $z = 1.29$ .

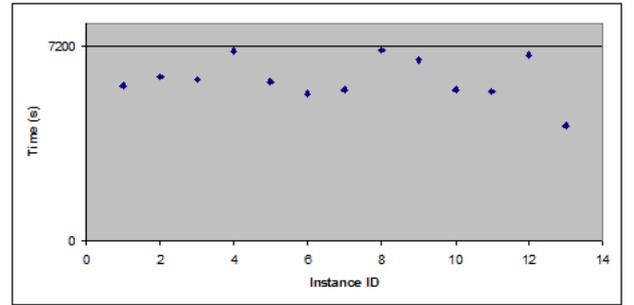
Then,  $D = f(x)(1 + a)$ , where  $a = 1.29\sigma_X + \mu_X$ . For our residuals, we get  $a = 1.525$ . This means, that in order to have a 10% chance of missing the deadline  $D$ , we need to choose  $x$  such that  $f(x) = \frac{D}{1+a}$ . For  $D = 3600$ , we should lower the deadline to  $D_1 = 3124$  and for  $D = 7200$ , we should lower the deadline to  $D_1 = 6247$ .



**Figure 14: POS tagging scheduling for adjusted  $D=3124$ , model (4)**

The results for the adjusted deadlines are given in figures 14 and 15 respectively. The result for the original deadline of 1 hour, show that we miss the deadline fewer times than in figure 12, but pay for an equivalent 30 instance hours of computation, which happens to be a worse fit than when using the first model and consuming 27 instance hours only.

The results for the deadline of 2 hours show that we are no longer missing the deadline and require 26 instance hours of computation. Without the adjusted deadline (figure 13) we require the same number of instance hours, but miss the deadline. Both solutions are better than those predicted by the first linear model (figure 11) which demands for 28 instance hours of computation.



**Figure 15: POS tagging scheduling for adjusted  $D=6247$ , model (4)**

**Table 2: Language complexity impact on POS tagging execution time**

Text	Size	# words	Wall time(min:s)
Dubliners	370 KB	67496	6:31.94
Agnes Grey	374 KB	67755	3:47.69

Based on the calculation above, a general good strategy can then be the following. For an initial deadline  $D$ , determine the minimum needed instances as  $\lceil \frac{V}{V_D} \rceil = i$ . If we are to spread the data approximately uniformly over  $i$  instances, we would give each at least  $\lceil \frac{V}{i} \rceil = V_{D_1}$ . The volume  $V_{D_1}$  leads to  $f(V_{D_1}) = D_1$ . If the adjusted deadline that guarantees a 10% chance to miss  $D$ , i.e.  $\frac{D}{1+a}$  is higher than  $D_1$ , we are satisfied with distributing the data into  $V_{D_1}$  bins over  $i$  instances. Otherwise, we will schedule for the adjusted deadline  $\frac{D}{1+a}$ .

Another experiment highlights the performance variability of POS tagging for texts of similar size, but different language complexity. We choose the *Dubliners* novel by James Joyce and *Agnes Grey* by Emily Brönte available from the Gutenberg project [2]. The experiment was repeated 5 times and the average wall time is shown. The results are summarized in Table 2.

For our news data set we do not see a dramatic improvement in the predictive power of our model derived by using random sampling. This can be expected of corpora that are uniform in terms of language complexity (average sentence length is an important parameter for POS tagging). For other corpora, as seen in the experiment above, random sampling can be vital to help capture the variation in text complexity.

## 6. RELATED WORK

A considerable amount of recent work focuses on investigating different aspects of commercial clouds: the quality of service received by users, the performance stability of the environment, the performance-costs tradeoffs of running different classes of applications in the cloud.

[17] and [10] investigate the effectiveness of constructing virtual clusters from Amazon EC2 instances for high-performance computing. [17] relies on standard HPC benchmarks that are CPU intensive (NAS Parallel Benchmarks) or commu-

nication intensive (mpptest) to compare the performance of virtual clusters of EC2 instances to a real HPC cluster. [10] performs a similar comparison using a real life memory and CPU intensive bioinformatics application (wcd). Both authors conclude that large EC2 instances fair well for CPU intensive tasks and suffer performance losses for MPI jobs that involve much communication over less efficient interconnects.

There is a lot of work that evaluates Amazon's S3 [9, 14] performance and cost effectiveness for storing application data. There is little literature on the usage and performance of EBS volumes for large scale applications.

Deelman et al [5] consider the I/O-bound Montage astronomy application and uses simulation to assess the cost vs performance tradeoffs of different execution and resource provisioning plans. One of the goals of their work is to answer a question similar to ours by finding the best number of provisioned instances and storage schemes to obtain a cost effective schedule. Their simulations do not take into account the performance differences among different instances and the flat rate per hour and partial hour Amazon pricing scheme which discourages having an excessively large number of instances that run for partial hours.

Other work by Juve et al [11] builds on [5] to address the more general question of running scientific workflow applications on EC2. They consider Montage as an I/O intensive application, and two other applications that are memory bound and CPU bound respectively and contrast the performance and costs of running them in the cloud with running on a typical HPC system with or without using a high performance parallel file system (Lustre). They note that I/O bound applications suffer from the absence of a high performance parallel file system, while memory-intensive and CPU-intensive applications exhibit similar performance. Their experiments are isolated to a single EC2 instance.

Wang and Ng [18] note the effect of virtualization on network performance, especially when the virtual machines involved are small instances that only get at most 50% of the physical CPU. They conclude that processor sharing and virtualization cause large network throughput and delay variations that can impact many applications.

Dejun et al [6] analyze the efficacy of using Amazon EC2 for service oriented applications that need to perform reliable resource provisioning in order to maintain user service level agreements. They find that small instances are relatively stable over time, but different instances can exhibit performance of up to 4 times from each other, which complicates provisioning.

## 7. FUTURE WORK

On the performance modeling side, we would like to explore the improvements of using more complex statistics tools to improve the accuracy of our predictions. We may use weighted curve fitting to obtain closer fits at larger volumes and allow for looser fits at smaller values since the corresponding measurements are also less stable.

We may also use performance measurements from instances

of different quality and take into account the likelihood of receiving such instances when devising an execution plan. For applications that use local storage, we may decide to invest in lightweight tests to establish the quality of the instances and then use different predictors for each instance quality level to decide how much data to send to meet the deadline.

We can also monitor application performance during execution and make dynamic scheduling decisions. If we find unresponsive instances, we force their termination and reassign their task to another instance. If we find that the application performance is not satisfactory, depending on the severity we can decide to terminate the instance and resume its task on a new instance or decide to let the instance run up to close to a full hour and move the rest of the work to another instance. Using EBS volumes makes dynamic adaptation easier. We can detach a volume from a poorly performing instance and resume work with another instance without explicit data transfers.

A direction for our future research is also to devise good execution plans for more complex workflows arising in text processing. We can schedule such workflows while making sure we assign full hour subdeadlines to groups of tasks ([19]). We plan to further explore data management possibilities for different classes of text applications we handle.

## 8. REFERENCES

- [1] Bonnie++. <http://www.coker.com.au/bonnie++/>
- [2] Project gutenberg. <http://www.gutenberg.org/>
- [3] S. Barker and P. Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of MMSys 2010*, February 2010.
- [4] J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd. Performance modelling of parallel and distributed computing using pazel. *IEEE International Performance Computing and Communications Conference, IPCCC-2000*, pages 485–492, February 2000.
- [5] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: the montage example. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.
- [6] J. Dejun, G. Pierre, and C.-H. Chi. EC2 performance analysis for resource provisioning of service-oriented applications. In *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing*, Nov. 2009.
- [7] K. C. et al. New grid scheduling and rescheduling methods in the grads project. In *in Proceedings of NSF Next Generation Software Workshop: International Parallel and Distributed Processing Symposium. Santa Fe, USA: IEEE CS*, pages 209–229. Press, 2004.
- [8] I. T. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. *CoRR*, abs/0901.0131, 2009.
- [9] S. L. Garfinkel. An evaluation of amazon's grid computing services: Ec2, s3 and sqs. Technical Report TR-08-07, Computer Science Group, Harvard University, 2008.

- [10] S. Hazelhurst. Scientific computing using virtual high-performance computing: a case study using the amazon elastic computing cloud. In *SAICSIT '08: Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries*, pages 94–103, New York, NY, USA, 2008. ACM.
- [11] G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B. P. Berman, and P. Maechling. Scientific workflow applications on amazon ec2. In *Workshop on Cloud-based Services and Applications in conjunction with 5th IEEE International Conference on e-Science (e-Science 2009)*, 2009.
- [12] D. Murray and S. Hand. Nephology towards a scientific method for cloud computing. In *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, April 2009.
- [13] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. Pace—a toolset for the performance prediction of parallel and distributed systems. *Int. J. High Perform. Comput. Appl.*, 14(3):228–251, 2000.
- [14] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon s3 for science grids: a viable solution? In *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64, New York, NY, USA, 2008. ACM.
- [15] W. Smith, I. T. Foster, and V. E. Taylor. Predicting application run times using historical information. In *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 122–142, London, UK, 1998. Springer-Verlag.
- [16] Stanford part-of-speech tagger.  
<http://nlp.stanford.edu/software/tagger.shtml>
- [17] E. Walker. Benchmarking amazon ec2 for high-performance scientific computing. *USENIX Login*, 33(5):18–23, 2008.
- [18] G. Wang and T. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing*, 2010.
- [19] J. Yu, R. Buyya, and C. K. Tham. Cost-based scheduling of scientific workflow application on utility grids. In *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*, pages 140–147, Washington, DC, USA, 2005. IEEE Computer Society.