Woodman S, Hiden H, Watson P.

Workflow Provenance: An Analysis of Long Term Storage Costs.

***In: WORKS '15 Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science*. 2015, Austin, Texas: ACM.**

Workflow Provenance: An analysis of long term storage costs

Simon Woodman
Dept. Computing Science
Newcastle University
Newcastle upon Tyne
simon.woodman@ncl.ac.uk

Hugo Hiden
Dept. Computing Science
Newcastle University
Newcastle upon Tyne
hugo.hiden@ncl.ac.uk

Paul Watson
Dept. Computing Science
Newcastle University
Newcastle upon Tyne
paul.watson@ncl.ac.uk

ABSTRACT

The storage and retrieval of provenance is a critical piece of functionality for many data processing systems. There are numerous cases where, in order to satisfy regulatory requirements (such as drug development and medical data processing), accurately reproduce results (scientific research) or to maintain financial transparency (for example to meet Sarbanes Oxley regulations in the US), a full and accurate provenance trace is vital.

Whilst it is always possible to meet these requirements by storing every piece of intermediate data generated by a sequence of calculations, the costs associated with retaining data that may have a low probability of future retrieval is significant. There is, however, an opportunity for a reduction in the cost of storage by opting not to store certain intermediate results that can be regenerated given a knowledge of the processing code and input data that generated them.

This paper presents a approach which is able, via a collection of past performance and provenance data, to make decisions based on the underlying storage and computation costs as to which intermediate data to retain and which to regenerate on demand.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

1. INTRODUCTION

When designing a system to process data and capture provenance, one obvious solution is to retain all intermediate data in a repository that expands each time any new work is performed - an approach which has been demonstrated in the Taverna workflow system [1]. This approach is guaranteed to work as one has confidence that any intermediate result will always be immediately available. The evident drawback, however, is that such a provenance store has an unbounded size that continuously grows regardless of whether the cost of retaining a piece of intermediate data exceeds the cost of re-generating it. If, however, a complete record of all of the code used to process data and any associated settings and parameters is stored, in many cases it will be possible to re-generate any piece of data using exactly the same process that was deployed in the original processing work. The drawback of this approach is that sometimes the computational cost required to regenerate results may exceed the cost of merely storing these results.

In order to effectively provide a comprehensive provenance storage system which can strike a balance between retaining and recalculating intermediate results, a cost model which can incorporate knowledge of storage-vs-computation costs is required. The decisions as to what to retain and what to re-generate are therefore a trade-off in terms of the long term storage cost and the cost (in terms of computation time) for re-generation for each given piece of data.

In this paper we will discuss which items of data can be regenerated on demand and show how a set of candidate policies can be created which allow the complete dataset to be regenerated if required. We show which data items must be retained, for example if there is insufficient knowledge to recreate them and use these rules to prune the set of candidate policies into valid ones. With a set of valid candidate policies, we are able to model the storage and compute resource requirement and then generate a financial cost model based upon a commercial cloud provider. These costs can then be projected into the future to indicate the overall cost of storing the data over a user defined period of time and to make decisions as to which pieces of data to keep and which to regenerate in order to minimise the financial impact of providing a comprehensive data retrieval capability.

2. PRIOR WORK

Research on provenance capture, storage and visualisation, particularly in the area of e-Science has been an active area for many years [2]. Much work has been done on how provenance can be captured from heterogenous systems such as workflow systems and databases and subsequently integrated and reasoned upon [3, 4]. Many early systems for provenance representation have roots in the Semantic Web movement and are based on RDF and related technologies [5, 6]. More recently the growth of non-relational, and par-

ticularly graph databases have been adopted as provenance stores given that a provenance trace is, inherently, a graph structure [7]. Such databases offer both a natural domain fit for the storage of provenance and powerful query interfaces to interrogate and interpret provenance traces.

Standards such as OPM [8] and its successor PROV [9] have been proposed and adopted by various systems in different ways. The ProvONE model [10] is of particular interest in this paper as it addresses the inadequacies [10] identified in the OPM standard and the base PROV model - namely that they are too generic to be directly useful within a scientific workflow system. Our previous work extended or 'subclassed' the generic OPM model to provide constructs useful in scientific workflow systems. ProvONE has introduced such concepts (such as `data` and `process` rather than the abstract `entity` and `activity`) natively, which is why we have adopted it in this work.

Typically systems which capture provenance do not attempt to also capture the raw data itself. PBase [11], which is based on the ProvONE model for interoperability, offers a web UI for user query and interaction. Early versions were based on an RDF store implemented in the Apache Jena framework but more recent versions have used the Neo4j graph database[12, 13]. The techniques presented in this paper could be applied to query the PBase system in order to optimise a third party data store.

One of the motivations for this work is to analyse the likelihood of being able to re-execute a given workflow at a future point. The authors of [4] show that a large percentage of scientific workflows decay over time rendering them unusable even to the original author. Systems able to capture the intermediate data and automatically reconstruct a workflow from provenance traces [7] help to overcome this.

The Chimera Grid based provenance capture system has some similar features to the technique being proposed in this paper. Chimera proposes a 'Virtual Data Grid' whereby original datasets are stored within distributed Grid storage [14]. Then a series of transformations are applied to them in order to create 'Virtual Data Sets'. These Virtual Data Sets can be reconstructed based on the provenance of their creation (deltas from the original dataset). The Chimera system can then estimate the cost of regenerating the dataset (e.g. a new replica) [15]. The difference between the Chimera system and that proposed here is that Chimera does not use this cost estimate to modify the storage policies in place for the Virtual Dataset. Instead they are used for making resource scheduling decisions when recreating data. Another difference to our work is our generation of multiple candidates policies and the application of Cloud cost models and different optimisation strategies.

There are also parallels between our work and that of Version Control Systems (VCS) such as Git and Subversion[16, 17]. For instance, VCSs must decide (in practice, at design time) whether to store every version of every document committed to them or just the steps needed to recreate each version. Storing data in this way would give very fast access to any revision of any document but would come at a very high storage cost. Generally, VCSs will store the latest revision (HEAD) and then reverse deltas necessary to recreate revision $HEAD-1$. Therefore to retrieve a previous version the VCS must apply the deltas from HEAD to the desired version in reverse each time it is requested but this gives a computation overhead. Given that in a normal development cycle most of the requests for previous revisions will be only a few before the HEAD revision this trade off makes sense. One can think of these two strategies as the extreme candidate sets that would be generated by technique presented in this paper.

Provenance aware storage systems such as PASS [18, 19] are interesting because they maintain both the data itself and the provenance trace for the generation of the data. One of the novel features of PASS is that it is able to generate executable scripts which describe the generation of a piece of data. These can then be applied to other data and the user can be sure that the same process has been followed. However they do not consider using the provenance and performance characteristics to optimise the cost of storing data.

3. DATA RETENTION POLICIES

In determining the various provenance data storage options, it is helpful to consider a simple data processing workflow (Figure 1). In this example, two activities operate sequentially on an entity E_0. Each activity operates on the results of the previous one such that, for example, $E_1 = A_0(E_0)$.

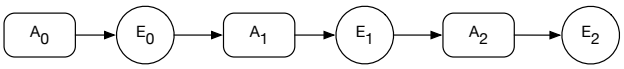

Figure 1: Simple data processing workflow

In the model presented in this contribution, sequences of data processing operations are performed by a workflow engine (in this case, e-Science Central [20], although other workflow engines could equally have been used to generate provenance traces such as Taverna [1] or Knime [21]). Workflows are commonly used to perform repetitive and complex data processing tasks and their very nature (large numbers of connected services) tends to produce extensive provenance traces [7]. In relating workflow terminology to that of the PROV model, we refer to the processing component of a workflow as *services* and consider a service to be a PROV **activity**. This **activity** is **associated** with the workflow engine co-ordinating the overall process. We also refer to PROV **entities** as *data*, such that services operate on data as opposed to activities operating on entities. This notation better reflects the fact that the applications and examples described in this paper were generated using a workflow system whilst still being compatible with current models of provenance described in Section 2.

3.1 Provenance storage options

In Figure 1, there are three pieces of data that the provenance storage system could at some point in the future be require to produce: E_0, E_1 and E_2. A simple strategy would be to simply store these pieces of data in perpetuity. However, given a system that captures the execution of services A_0, A_1 and A_2 and all of the associated metadata for these activities (code version algorithm settings, etc), it is possible

to use this information to regenerate an exact copy of any piece of data on demand. In generating any piece of intermediate data, it is also necessary to account for the cost/time for the regeneration of any upstream data which may have been deleted. For example, in this scenario, upstream data for the entity, E_1 would be E_0.

When considering which pieces of data to retain in the provenance store and which to discard in favour of regeneration, for the example presented in Figure 1, there are a number of candidate options (Table 1) which refer to the various storage / regeneration options.

Table 1: Candidate Set for Simple Workflow Example

	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
E_0	K	K	K	K	R	R	R	R
E_1	K	K	R	R	K	K	R	R
E_2	K	R	K	R	K	R	K	R

In Table 1, **K** represents keeping the piece of intermediate in the provenance repository, whilst **R** represents opting to regenerate that piece of data. For this simple example, therefore, there are eight candidate options (C_0 - C_7) representing all of the various provenance storage solutions. In general, the number of storage options is 2^n, where 'n' is the number of services operating within the workflow. In reality, however, some of these candidate options are not feasible. There are a number of reasons for this:

Input data In some cases, it is not possible to regenerate data. This occurs for example when some data is created by a process that is not under the control of the workflow engine. Typically, this occurs when data is uploaded by a user. We consider data in the system that doesn't have any incoming provenance as such an edge case.

Non deterministic operations When making a decision to discard a piece of data, the assumption is that re-running the service that generated it with identical input data and settings will result in an exact copy of that piece of intermediate data. This assumption, however, may not always be valid. In our model, we refer to such operation as non-deterministic with the result that any intermediate data generated by non-deterministic services must always be retained in the provenance storage system. Examples of this include machine-learning operations that have a stochastic element to them, such that different runs on the same piece of input data will produce subtly different results[22]. Another type of non-deterministic service is one that retrieves data from an external source - this source could not be guaranteed to deliver the same data each time it was queried and so would be considered non-deterministic.

Non idempotent operations Some services have the potential to modify state in an associated system. Clearly when these services are present it is not an option to re-run them in order to regenerate their output data as this may make unwanted changes to attached systems. Again, in these cases, data generated must be retained in the provenance storage system. An example of this type of service would be one that wrote data into an external database.

In the example above, if A_0, the activity involved in the creation of E_0, was a user uploading data to the system, E_0 would not be eligible for regeneration, removing candidates: $C_{4,5,6\&7}$ from Table 1. In general, the number of valid candidate solutions to the problem of optimally storing provenance for a given workflow is:

$$N = 2^{n-(i+d+m)} \quad (1)$$

where:

n is the total number of services

i is the number of input data services

d is the number of non-deterministic services

m is the number of non-idempotent services

The number of valid candidate policies is expressed in terms of the number of activities in the graph rather than the number of entities. This is because in order to regenerate an entity the activity must be re-executed. If an activity generates multiple entities they will all be regenerated but it is the individual entity that we are interested in. Candidate policies which do not keep all of the outputs of an activity are still valid - the entities may be of unequal size and so have different implications for storing them.

In making the decision of which of the candidate solutions to adopt, we analyse the respective costs of regenerating -vs- storing provenance data for a typical Cloud provider (Section 5).

4. STORING AND REGENERATING ENTITIES

Given a set of valid candidate retention policies for the data (entities) generated as part of the workflow execution we can compute storage and compute requirements associated with adopting each retention policy. This "logical" cost will be mapped onto a monetary cost in Section 5.

The following must be considered when calculating the logical cost of adopting each policy:

Storage Costs The cost of storing an entity in Cloud storage such as Amazon S3 or Azure Blob Store for one month.

Regeneration Costs the cost of the compute time to enact the activities necessary to regenerate the entity. This will be expressed in terms of CPU hours (CPUh)

Fixed Costs The costs of maintaining any underpinning systems such as application servers, databases, workflow engines. We will ignore this cost as it is assumed to be fixed across all retention policies.

4.1 Regeneration Process

When calculating the logical cost of each candidate policy there are different ways that we could calculate the cost of adopting each policy, ie. the resources required to regenerate the required entities. These differ whether they consider the system as a whole or treat each piece of data independently.

4.1.1 Global Regeneration Cost (GRC)

The GRC policy calculates the cost of returning the system to a previous state where all entities are available rather than regenerated. As such, it is equivalent to the C_0 retention policy shown in Table 1 where everything is kept.

The GRC policy is straight forward and assumes a linear recreation of the entities which need to be regenerated. Given that everything was created in the first place and then either kept or marked for regeneration we ensure that, by running everything in order, the inputs for activity $n+1$ are always available after activity n has run. Thus the cost of regenerating the policy is simply the sum of the execution time of all activities whose output has not been kept:

$$compute_time = \sum_{i=0}^{j} execution\_time(A_i) \quad (2)$$

where A_j is in the set of activities which produce entities which need to be regenerated.

Evidently, we must also include the cost of storing those entities which are not being regenerated. This is expressed below where there are k entities which the policy decides to keep rather than regenerate:

$$storage_volume = \sum_{i=0}^{k} storage\_cost(E_i) \quad (3)$$

The resources required to implement the GRC policy is simply the storage volume + the compute time:

$$policy\_resources = compute_time + storage_volume \quad (4)$$

4.1.2 Entity Re-creation Cost (ERC)

Whilst the GRC policy calculates the cost of regenerating all the entities which were previously present, the ERC policy calculates the resources required to regenerate each of the entities independently. Although this is a more complex scenario, we believe it more accurately expresses the real world where single entities can be requested independently.

In order to regenerate an entity, the system will need to execute the activities between the 'last' available entity and the desired entity. By 'last' available entity we mean the closest 'upstream' entity in the provenance graph. Without branches, this is the sum of the execution times between an entity, E_n and the last stored entity, E_j.

$$compute\_time_n = \sum_{i=j}^{n} execution\_time(A_i) \quad (5)$$

Given a more complex structure where branches exist, the equation shown above can be generalised into one which includes the cost of regenerating the entities for the activities along the different branches involved in the generating of entity E_n. We define *nda* as the number of distinct activities along the branches leading to E_n.

$$compute\_time_n = \sum_{i=0}^{nda} execution\_time(A_i) \quad (6)$$

As before, we must include the storage impact of retaining k entities which are being kept rather than regenerated.

$$storage\_volume = \sum_{i=0}^{k} storage\_cost(E_i) \quad (7)$$

The total resource requirement is the sum of the compute time to regenerate each of the entities which have not been kept and the storage volume of those which have, where there are p entities which need to be regenerated

$$policy\_cost = (\sum_{i=0}^{p} compute\_time_p) + storage\_volume \quad (8)$$

For the remainder of this paper, we will use the ERC policy due to its more accurate representation of the costs of regenerating individual items.

4.2 Example

Let us consider the set of candidate retention policies described in Section 3.1 which relate to the example shown in Figure 1. In order to calculate the storage and regeneration costs we must first determine the size of the entities and the duration of the activities. This could be collected by hand or automatically by the Workflow Engine used to coordinate the activities[23].

For this example we have based the data volumes and activity durations on a real world workflow which was used to calculate location from Bluetooth Low Energy beacons. The actual workflow deployed in the study was more complex, but has been simplified here for readability. The activities omitted do not affect the outcome and are largely concerned with extracting metadata from the headers and integrating into the execution platform [20]. The durations and data volumes are shown in Table 2.

The salient points in this example are that the distance calculation generates about 2x the data volume and both this activity and the filtering are *relatively* compute intensive

	Desc	Duration (min)	File Size (GB)
A_0	Download Data	2	
E_0	RSSI from Beacons		0.05
A_1	Calculate distance from RSSI	5	
E_1	RSSI + Distance		0.1
A_2	Filter and visualise	5	
E_2	Report/Graphs		0.001

Table 2: Example Durations and Data volumes

based on the volume of input data. In order to calculate the costs of the different candidate policies we use the equations shown in Section 4.1.2. For instance for C_3, where the policy is `KRR`, we must calculate the storage and compute costs. The storage costs are just for E_0 and therefore 0.05GB. The compute costs are the cost for regenerating E_1 and for regenerating E_2. The cost of E_1 has no upstream computations which need to be included so it is 0.08h. For E_2 we must also include the cost of regenerating E_1 again as it is an upstream requirement, therefore it is $0.08h + 0.08h = 0.16h$. Thus the total compute cost for C_0 is $0.08h + 0.16h = 0.24h$. The full costs for each of the policies is shown in Table 3.

In order to generate the data for the example shown above and produce the cost models for the remaining examples in this paper, we created a simple simulation of a workflow using a JavaScript representation[1] of a set of simple interconnected services executed within the Node.js environemnt [24]. Each service in this model has an output data volume and computation time associated with it. The graph structure of this model, coupled with the actual resource requirements measured during the workflow execution, is used to calculate the upstream regeneration costs for each service when computing the ERC costs.

	Strategy	Storage (GB)	Compute (h)
C_0	KKK	0.151	0.000
C_1	KKR	0.150	0.083
C_2	KRK	0.051	0.083
C_3	KRR	0.050	0.240

Table 3: Logical Costs of the ERC Policy

5. COST MODELS

The previous Section demonstrated how we are able to generate a logical cost for each candidate retention policy based on the amount of compute time and storage required by that policy. This section will show how we can apply cost models from cloud computing providers to assign a monetary cost to each candidate in order to produce a ranked set which can be used to select a desirable provenance storage strategy.

Cloud computing providers offer a variety of different specifications of hardware in terms of the available CPU, GPU, RAM and I/O performance. These specifications vary in the amount that they cost to operate per hour. Individual applications will have minimum requirements for them to be able to execute. For instance, the PAC1 example presented in Section 5.2 requires at least 10GB of physical memory. The characteristics of applications when more resources are available will differ from application to application. Applications which are multi-threaded may be able to make use of additional CPU resources whereas those with a single thread are unlikely to see a performance improvement even if additional compute resources were made available.

As we are unable to generalise the requirements of applications, we favour generating figures which can act as a guide to indicate the cost if different machine types were required to run the application under each candidate policy. Figure 2 shows the costs for retaining all of the data vs using 3 different machine types from the Amazon EC2 cloud[2]: t2.large (2 CPU, 8GB memory), m4.large(4 CPU 16 GB memory), m4.2xlarge (8 CPU, 32GB memory).

In addition to multiple machine types, cloud providers also offer different types of storage with different characteristics. For instance, Amazon offers S3 - highly available redundant storage, a version that has reduced redundancy (but is still available on-demand) and Glacier which offers higher latency access (up to 24 hours to retrieve data). These storage offerings decrease in price as the redundancy decreases and the latency of retrieving the data increases. The current costs[3] for the relevant Amazon Web Services components are shown in Table 4.

Service	Cost (US$)
t2.large	0.104 CPU/h
m4.xlarge	0.252 CPU/h
m4.2xlarge	0.504 CPU/h
S3	0.03 GB/m
S3 RR	0.024 Gb/m
Glacier	0.01 GB/m

Table 4: AWS Component Costs

Figure 2 shows the relative costs of the different machine types and storage options for the KRR strategy compared with the policy that retains all of the provenance data in S3 (i.e. a KKK strategy). This result demonstrates that the selection of machine type has a significant impact on the optimum provenance storage strategy. In this case, selecting a large machine (m4.2xlarge) means that for 40 months, the optimum strategy is to retain all intermediate data instead of regenerating it. If the application is able to run on a smaller machine the cross over point is earlier - around 9 months for a t2.large machine. In many cases, the smallest usable machine type will be dictated by the requirements of the data processing operations and will not be changeable and this constraint will have a significant impact on the selected data retention strategy.

Once we determine a suitable size of machine and storage option for an application we can calculate the cost for implementing each strategy over a period of time. Figure 3 shows the four different candidate policies for the example presented earlier plotted over a three year period. We can

[1]This model and the configurations needed to regenerate the results presented in this paper is available at https://bitbucket.org/hghid/provenance-model

[2]http://aws.amazon.com/ec2/instance-types/

[3]As of summer 2015

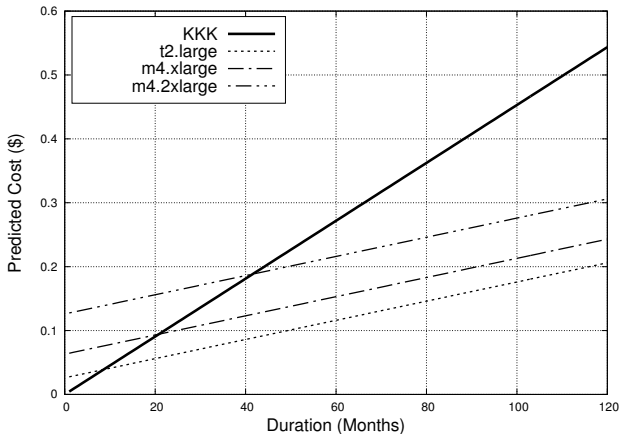

Figure 2: Effect of machine size on cost of KRR

see that initially it is optimal to keep all of the data therefore, for the first 7 months, the KKK option is the cheapest. However, if we wish to retain the data for a longer period of time, the KRK policy becomes cheaper and will remain so. The *shape* of the process (i.e. the relative data sizes and compute times observed as the process executes) dictates that it is expensive to keep the intermediate data for a long period if time and better to regenerate it on demand.

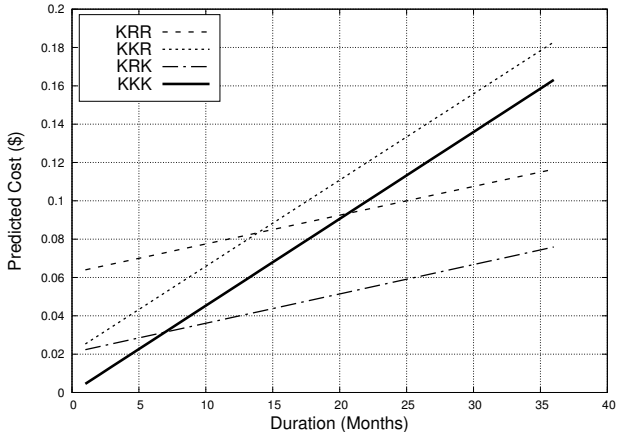

Figure 3: Cost of Retention policies over 3 years

When considering the monetary cost of storying data clearly the duration will have a large impact on the retention strategy chosen. Many UK funding authorities currently stipulate that data generated within a research project that they fund must be retained for 10 years since the date of last access (EPSRC, MRC, BBSRC, Wellcome Trust). There are sectors such as industrial science and pharmaceuticals where data must be retained much longer and potentially can never be deleted.

Figure 4 shows that the impact of the different strategies is even more marked when looking at a longer duration for keeping the data (10 years in this example). The KRK policy is approximately 50% of the cost of the KKK policy. The overall costs of each policy is shown in Table 5 and corresponds to the logical costs presented in Table 3.

	Strategy	Storage ($)	Compute ($)	Total ($)
C_0	KRR	0.180	0.063	0.242
C_1	KKR	0.540	0.021	0.561
C_2	KRK	0.184	0.021	0.204
C_3	KKK	0.544	0.000	0.544

Table 5: Example Strategy Costs

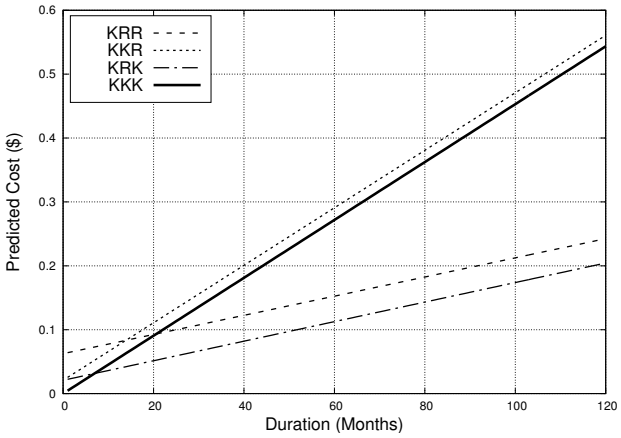

Figure 4: Cost of Retention policies over 10 years

Although the numbers in this example are small (even the most expensive policy will only cost $0.5 to store the data for a 10 year period), it should be noted that this is for a single execution of a single workflow. It is not uncommon for both the data sizes to be significantly larger than this (as described in the following Section) and for there to be many thousands of executions of the workflow [25] resulting in significant cost implications.

5.1 Modelling future Storage costs

Our assumption so far has been that the cost of storing data in a cloud based system such as Amazon S3 remains constant over time. Whilst Kryder's Rate[26] appears to be slowing and the original prediction unlikely to be achieved, storage, as with most other computer components, is becoming cheaper to produce and is able to store information at higher densities.

When we are projecting costs 10 years into the future the potential storage costs can have a significant impact on the overall price of each retention policy. Thus our use of a constant rate would appear a little naïve. In order to adjust for this we have looked at historical price data for the Amazon S3 service (Glacier has remained a constant $0.01/GB/m since introduction and S3 RR historical pricing is not available). The cost of S3 is shown in Figure 5 and shows the price dropping over time. Notably, the price as of Summer 2015 is 1/5 of the cost when it was first introduced.

Whilst it is possible to model the precipitous price drop shown in Figure 5, it is difficult to believe that the storage cost will indeed drop to zero in the imminent future. AWS costs have dropped in steps over the past decade in response to both dropping hardware costs and competitive pressures.

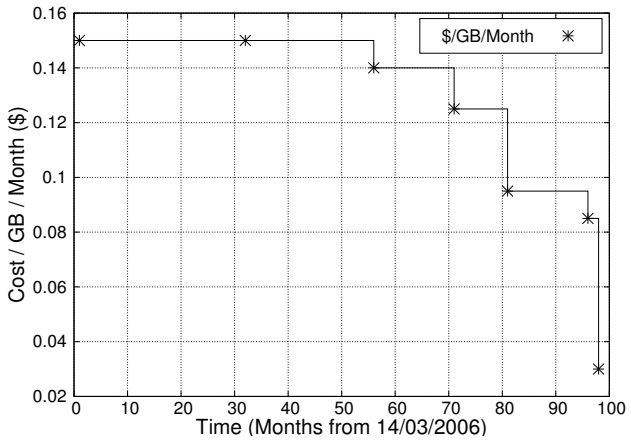

Figure 5: Historical costs of Amazon S3

Although there will undoubtedly be more step-wise reductions in storage costs in the future, it is impossible to include this in any future projections. On average, however, AWS storage costs since 2006 have dropped at 1.6% per month. This drop can be included in the cost projections shown for the example above, the effect of which is shown in Figure 6.

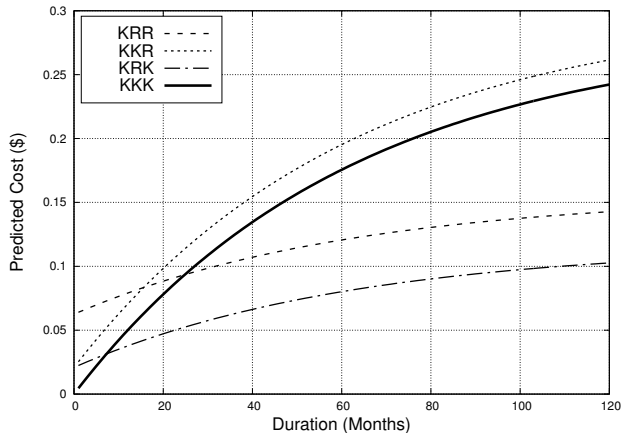

Figure 6: Future cost projection with monthly drop

Although Figure 6 shows a drop in long term costs, the overall strategy for lowest cost (KRK) remains unchanged.

5.2 PAC1 Example

The aim of the example presented in Section 3 of this paper was to be simple enough to demonstrate our approach to data storage optimisation by illustrating the process of candidate policy generation, pruning, and the application of different cost models to each policy. This section will introduce a larger example based on a real world workflow application. The application was used to analyse physical activity using accelerometers as part of a study of over 1000 participants in the North East of England[27]. The workflow used in this study employed the PAC1 algorithm to classify physical activity into one of four levels at a given point in time[28]. This was then used to construct a report showing the percentages of time that study participants spent at various activity levels over a two week period.

The structure of the workflow used to process the data is shown in Figure 7 and comprises 9 activities. Initially the data for one participant is downloaded onto the Workflow Engine and then converted to a CSV format. The original storage is a binary compressed format but the PAC1 implementation used in this case operates on uncompressed data. Once uncompressed, the algorithm (which is written in Octave) is executed and a report is generated. The headers are extracted from the original file and attached as metadata to the results, which are stored in e-Science Central (the system being used to coordinate the workflow execution). Finally the report is saved and some other housekeeping tasks are performed to index the results. Two of the salient points of this process are that the conversion to CSV produces a large file which is approximately 4.5x the size of the compressed input data, and the PAC1 algorithm is memory intensive, requiring approximately 10GB of physical RAM.

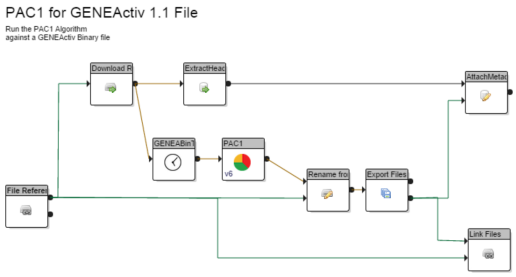

Figure 7: PAC1 Workflow

In this process, most activities are repeatable (as described in Section 3.1 but as with the example presented the first blocks, which deal with obtaining the input data, are not repeatable. Thus, we need to keep the output of Get and Download references. Additionally, the Link Files activity is non-idempotent. Following an execution of this workflow the provenance trace, shown in Figure 8, is generated.

There are 2^9 candidate retention policies for this provenance trace however, given the three activities mentioned above are not repeatable, this set can be pruned to 64 valid candidates where $K(E_0)$, $K(E_1)$, $K(E_2)$ and all other entities may be either `Keep` or `Regenerated`.

The durations of the various activities and data volumes generated are shown in Table 6.

When we apply the file size and duration data to the valid candidate policies we can see that keeping all of the data would cost $2 over a ten year period. The most expensive option is `KKKKRKRKK` at $2.6 so keeping all of the data is not the worst case scenario. However, the best case scenario for the ERC technique is `KKKRKRRRR` which will cost just under $0.5 to store and regenerate over a ten year period.

Looking at the *shape* of the workflow the numbers presented are not surprising. The policies which are more expensive favour regenerating the report and storing the large intermediate CSV file. The cheap policies (the distribution of policies is uneven with most costing around $0.5 or $2.5) favour the opposite, to keep the small report and regenerate the large CSV file if required, a more natural choice.

	Desc	Duration (min)	File Size (GB)
A_0	Get File Reference	0.05	
E_0	File Reference		0.0001
A_1	Download Ref	2	
E_1	Bin File		0.735
A_2	Extract Headers	1	
E_2	Header Info		0.0005
A_3	Convert to CSV	10	
E_3	CSV File		3.2
A_4	PAC1	30	
E_4	Report		0.001
A_5	Rename Report	0.2	
E_5	Report		0.001
A_6	Export Files	0.2	
E_6	Report Ref		0.0001
A_7	Attach Metadata	0.2	
A_8	Link Files	0.2	

Table 6: Performance Stats for PAC1 process

Further, whilst most of our calculations have assumed regenerating the data a single time, this example shows us that we could regenerate everything four times and it would still be cheaper than storing the entire dataset.

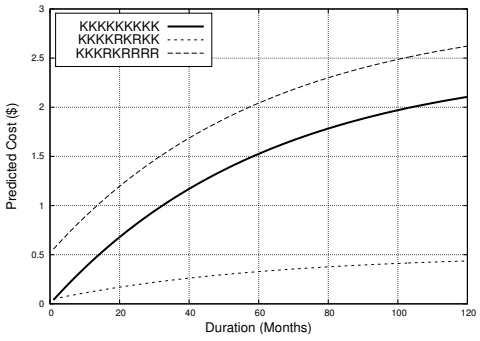

Figure 9: PAC1 Graph

6. FUTURE WORK

The number of candidate retention policies is generally 2^n where n is the number of activities in the chain of data processing tasks (i.e. the number of workflow blocks in this case). This number grows exponentially and dealing with more than moderate sized workflows will need a different approach for candidate generation. The binary keep/regenerate nature of the candidate options would appear to lend themselves well to optimisation with a Genetic Algorithm, which can operate on long bit strings representing extremely high dimensional search spaces. Such an approach may work well when the number of blocks in a workflow would preclude an exhaustive search of all of the candidate storage/regeneration options in a reasonable period of time.

So far we have not considered different ranking strategies for the costed candidate policies - implicitly we have been assuming that cheapest is best. However, it is possible to think of scenarios where an application may wish to optimise on multiple parameters instead of just cost. For instance, the latency of regenerating everything may not be acceptable and they may wish to limit the regeneration time of any entity to a certain maximum time limit. This could interplay with mixed storage policies where some items are stored in cheaper high latency storage and others in storage which is available more quickly.

7. CONCLUSIONS

This paper has demonstrated a method for determining the optimum cost of maintaining the data generated by a workflow over a given period of time. It has done this by comparing the monetary cost of retaining vs regenerating data files. The model presented considers a set of data processing activities to be a sequential list of operations which generate intermediate data of varying sizes. By developing a cost model encapsulating the trade-off between storage and regeneration and parameterising this model with typical Cloud provider costs (accurate as of 2015), we have shown it is possible to produce an optimum storage strategy. This has been demonstrated on a number of example data processing tasks, the performance data of which has been gathered from actual applications.

The algorithm used to calculate the optimum strategy considers all of the valid provenance storage policies and delivers its result in the form of a ranked list of candidate solutions. This allows a selection to be made based not just on minimum financial cost. For example, a decision may be made to favour recalculation where the additional cost of doing this is marginal if this was preferable to maintaining and backing up a large provenance store.

8. ACKNOWLEDGMENTS

This work has been funded as part of the SiDE project, the RCUK Digital Economy Research Hub on Social Inclusion through the Digital Economy, EP/G066019/1 and by the Digital Institute at Newcastle University.

9. REFERENCES

[1] Katherine Wolstencroft et al. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 2013.

[2] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005.

[3] J. Freire, D. Koop, E. Santos, and C.T. Silva. Provenance for computational tasks: A survey. *Computing in Science Engineering*, 10(3):11–21, May 2008.

[4] Khalid Belhajjame, Marco Roos, Esteban Garcia-Cuesta, Graham Klyne, Jun Zhao, David De Roure, Carole Goble, Jose Manuel Gomez-Perez, Kristina Hettne, and Aleix Garrido. Why workflows break: Understanding and combating decay in taverna workflows. In *Proceedings of the 2012 IEEE 8th International Conference on E-Science (e-Science)*, E-SCIENCE '12, pages 1–9, Washington, DC, USA, 2012. IEEE Computer Society.

[5] Jun Zhao, Carole Goble, Robert Stevens, and Sean Bechhofer. Semantically linking and browsing provenance logs for e-science. In *Semantics of a*

Networked World. Semantics for Grid Databases, volume 3226 of *Lecture Notes in Computer Science*, pages 158–176. Springer Berlin Heidelberg, 2004.

[6] J. Myers, C. Pancerella, C. Lansing, K. Schuchardt, and B. Didier. Multi-scale science: Supporting emerging practice with semantically derived provenance. In *CEUR Workshop*, 2003.

[7] Simon Woodman, Hugo Hiden, Paul Watson, and Paolo Missier. Achieving reproducibility by combining provenance with service and workflow versioning. In *Proceedings of the 6th workshop on Workflows in support of large-scale science*, WORKS '11, pages 127–136, New York, NY, USA, 2011. ACM.

[8] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van Den Bussche. The Open Provenance Model — Core Specification (v1.1). *Future Generation Computer Systems*, 7(21):743–756, 2011.

[9] Paul Groth and Luc Moreau. Prov-overview. http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/, 2013.

[10] Víctor Cuevas-Vicenttín et al. Provone: A prov extension data model for scientific workflow provenance. http://vcvcomputing.com/provone/provone.html, 2014.

[11] Víctor Cuevas-Vicenttín, Parisa Kianmajd, Bertram Ludäscher, Paolo Missier, Fernando Chirigati, Yaxing Wei, David Koop, and Saumen Dey. The PBase Scientific Workflow Provenance Repository. In *Procs. 9th International Digital Curation Conference*, volume 9, pages 28–38, San Francisco, CA, USA, oct 2014.

[12] Apache Software Foundation. Apache jena. https://jena.apache.org/.

[13] Neo4j. http://www.neo4j.org, September 2011.

[14] I. Foster, J. Vöckler, M. Wilde, and Yong Zhao. Chimera: a virtual data system for representing, querying, and automating data derivation. In *Scientific and Statistical Database Management, 2002. Proceedings. 14th International Conference on*, pages 37–46, 2002.

[15] Ian T Foster, Jens-S Vöckler, Michael Wilde, and Yong Zhao. The virtual data grid: A new model and architecture for data-intensive collaboration. *SSDBM*, 3:11–11, 2003.

[16] Software Freedom Conservancy. Git. https://git-scm.com/.

[17] Apache Software Foundation. Apache subversion. https://subversion.apache.org/.

[18] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. Provenance-aware storage systems. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 4–4, Berkeley, CA, USA, 2006. USENIX Association.

[19] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana Maclean, Daniel Margo, Margo Seltzer, and Robin Smogor. Layering in provenance systems. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, pages 10–10, Berkeley, CA, USA, 2009. USENIX Association.

[20] Hugo Hiden, Simon Woodman, Paul Watson, and Jacek Cala. Developing cloud applications using the e-science central platform. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1983), 2013.

[21] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. KNIME: The Konstanz Information Miner. In *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*. Springer, 2007.

[22] Paolo Missier, Simon Woodman, Hugo Hiden, and Paul Watson. Provenance and data differencing for workflow reproducibility analysis. *Concurrency and Computation: Practice and Experience*, 2013.

[23] Hugo Hiden, Simon Woodman, and Paul Watson. A framework for dynamically generating predictive models of workflow execution. In *Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science*, WORKS '13, pages 77–87, New York, NY, USA, 2013. ACM.

[24] Ryan Dahl. The node.js platform.

[25] Jacek Cala, Hugo Hiden, Simon Woodman, and Paul Watson. Cloud computing for fast prediction of chemical activity. *Future Generation Computer Systems*, 29(7):1860 – 1869, 2013.

[26] C. Walter. Kryder's Law. *Scientific American*, 293:32–33, 2005.

[27] Joanna Collerton, Karen Barrass, John Bond, Martin Eccles, Carol Jagger, Oliver James, Carmen Martin-Ruiz, Louise Robinson, Thomas von Zglinicki, and Tom Kirkwood. The newcastle 85+ study: biological, clinical and psychosocial factors associated with healthy ageing: study protocol. *BMC Geriatrics*, 7(1):14, 2007.

[28] S Zhang, AV Rowlands, P Murray, and TL Hurst. Physical activity classification using the genea wrist-worn accelerometer. *Medicine and Science in Sports and Exercise*, 44(4):742–748, April 2012.

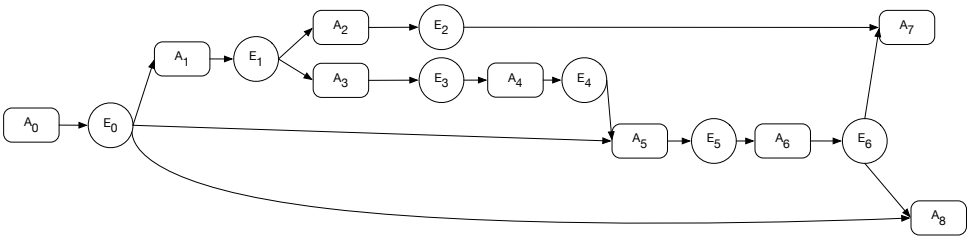

Figure 8: PAC1 Provenance Trace