Efficient Re-computation of Big Data Analytics Processes in the Presence of Changes: Computational Framework, Reference Architecture, and Applications

(Invited Paper)

Paolo Missier and Jacek Cała
School of Computing
Newcastle University
Newcastle upon Tyne, UK
Paolo.Missier@ncl.ac.uk, Jacek.Cala@ncl.ac.uk

***Abstract*—Insights generated from Big Data through analytics processes are often unstable over time and thus lose their value, as the analysis typically depends on elements that change and evolve dynamically. However, the cost of having to periodically "redo" computationally expensive data analytics is not normally taken into account when assessing the benefits of the outcomes. The `ReComp` project addresses the problem of efficiently re-computing, all or in part, outcomes from complex analytical processes in response to some of the changes that occur to process dependencies. While such dependencies may include application and system libraries, as well as the deployment environment, `ReComp` is focused exclusively on changes to reference datasets as well as to the original inputs. Our hypothesis is that an efficient re-computation strategy requires the ability to (i) observe and quantify data changes, (ii) estimate the impact of those changes on a population of prior outcomes, (iii) identify the minimal process fragments that can restore the currency of the impacted outcomes, and (iv) selectively drive their refresh. In this paper we present a generic framework that addresses these requirements, and show how it can be customised to operate on two case studies of very diverse domains, namely genomics and geosciences. We discuss lessons learnt and outline the next steps towards the `ReComp` vision.**

***Keywords*-process recomputation; recomputation optimisation; provenance; data analysis; black-box process; workflow**

I. Introduction

In many areas of science and business, knowledge insights produced using Big Data analytics are valuable, but often unstable over time. This is because the processes used to produce such knowledge outcomes are sensitive to multiple types of changes that may occur to the reference data sources, application and system libraries used by the process, as well as to the raw input data itself. In high throughput genomics, for instance, where the processing time for a batch of whole-genomes is measured in tens of HPC cluster hours [1], the pipelines depend both on algorithm packages that are periodically updated (e.g. GATK[1]), as well as on reference datasets that evolve rapidly over time (for instance, the well-known dbSNP variants database[2]).

[1]https://software.broadinstitute.org/gatk/
[2]https://www.ncbi.nlm.nih.gov/snp/

Failing to react to these changes leaves the clinician in the genetic diagnostic lab exposed to the risk of using obsolete results for important decision-making, while the genetics researcher will be missing out on the opportunity to improve their experimental results. On the other hand, over-reacting to each and every change is likely inefficient, as most changes may have little real impact on the original outcomes, in terms of updating a diagnosis or changing a business decision.

In this paper we address the problem of efficiently re-computing unstable analytics outcomes in response to changes in the elements that contributed to their original computation, namely inputs, reference datasets, tools, libraries, and deployment environment.

We suggest that an ideal strategy that avoids unnecessary re-computation requires the ability to (i) observe and quantify changes, (ii) estimate the impact of those changes on the original outcomes, and (iii) identify the process fragments that are dependent on the changed element. To formalise such a strategy, we have developed `ReComp`, a *meta-process* designed to harness an underlying analytics process P. `ReComp` accepts a sequence of input changes and provides the data analyst with do/don't re-compute decisions in response to each of those changes. `ReComp` is generic in that knowledge of (i) and (ii) must be provided for each type of underlying process P and for the specific types of data handled by P, and (iii) is computed using a description of P's structure, along with the recorded history of its past executions, i.e., the *provenance* of P's prior outcomes. Note that, for the purpose of this paper, we focus exclusively on changes in the reference datasets, while other types of changes are beyond the scope of this work and are still being investigated.

A. Paper contributions

Firstly, we provide a detailed account of the `ReComp` computational model as well as of the reference technical architecture of its domain-agnostic, customisable framework. Secondly, to illustrate `ReComp`'s customisation we have chosen two case studies taken from two very different domains of science and engineering, namely a simple process

for the interpretation of human genetic variants to help the diagnosis of genetic diseases, and a simulation of flood events in urban areas. For each of these two case studies, we provide a brief description of *data diff* and *impact estimation functions* for each of the two case studies.

Finally, we discuss lessons learnt from the customisation exercise. Realistically, often only imperfect knowledge of (i-iii) above will be available in practice, limiting the accuracy of the resulting re-computation decisions. In particular, a *black-box* process that does not reveal its internal structure makes it difficult to achieve (iii), namely identify the sub-process fragments that need re-computing. We report on the balance between the accuracy of data diff and impact estimation functions designed for a particular process, and the resulting accuracy of the resulting `ReComp` decisions.

B. Case studies

Our case studies provide a practical motivation for this work, by giving examples of two user processes. The first aims to control re-computation of a *black-box*, compute-intensive simulation process used in flood modelling. The second shows the use of `ReComp` to manage a data analytics workflow used to help the expert geneticist's interpretation of human variants for genetic diagnostic purposes. This is a simple *grey-box* process, that is, a workflow that only reveals its internal structure at the level of connections amongst its component workflow blocks. The two cases differ in structure of their processes, the computational complexity, data dependencies and in their ability to capture data and process provenance. They will help us show the flexibility in how `ReComp`, with little adaptation, may be used in these diverse environments.

1) Black-box process: City Catchment Analysis Toolkit (CityCAT) [2] is a modelling software developed at Newcastle University to simulate the risks of floods in the extreme weather conditions. It uses a coupled 1D/2D hydraulic model and requires extensive amount of processing power to generate a set of time frames that show water flowing through the selected urban area. Inputs to the simulation, like the buildings and infrastructure of an urban area, undergo continuous change, and so it is crucial to make a decision whether to save costs and rely on a past version of simulation outputs, or update the inputs and redo the simulation and flood risk analysis again.

2) Grey-box process: Simple Variant Interpretation (SVI) [3] is a tool that aims to identify variants in human genome that may be responsible for an individual's phenotype, i.e. the manifestation of a suspected genetic disease. Recently, it has been implemented as a workflow on e-Science Central (e-SC) – our in-house workflow management system (WfMS) [4], [5].

The SVI workflow relies on two external reference databases: one, called `OMIM GeneMap`, translates phenotype description into a broad set of related genes. The other, `NCBI ClinVar`, is used to determine whether a specific genetic variation may have deleterious effects on the patient's genes function. Using the databases, SVI takes annotated patient variants and suspected phenotype and classifies the variants according to a simple traffic light system of the red, green and amber colour to denote pathogenic, benign and variants of unknown or uncertain pathogenicity, respectively.

The reference databases are updated on a daily and monthly basis, respectively, and these changes may potentially affect a large cohort of patients' genomes analysed. Thus, the key question is when, if ever, a particular patient analysis should be rerun to ensure it reflects the most up-to-date knowledge of genetic mutations but without incurring unnecessary costs.

C. Overview of the approach

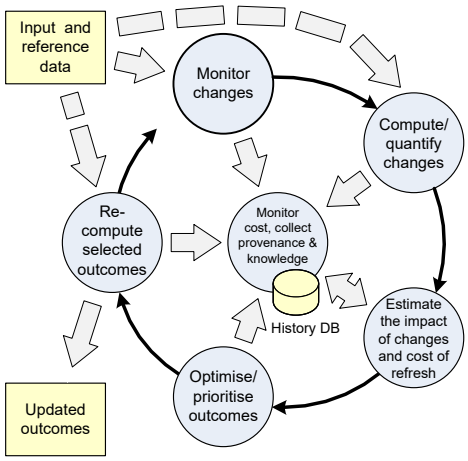

Figure 1. The main loop in the `ReComp` framework that handles selective re-computation of the user process.

`ReComp` defines a meta-process that is able to monitor and control an underlying data-intensive process P. A high-level functional view of such meta-process is shown in Fig. 1. Firstly, we assume that P's execution is provenance-aware, that is, a provenance trace is generated every time P is executed. `ReComp` acquires all such traces and stores them into its History Database (HDB). Note that traces can be either very granular or very high-level depending on the process execution environment. `ReComp` is tolerant to varying provenance detail, as explained in Sec. III-B, however a completely black-box process is going to be less amenable to optimisation.

`ReComp`'s input consists of notifications of changes that have occurred in any of P's data dependencies, namely reference datasets as well as inputs, which are provided by source-specific monitoring facilities (for instance, we assume that `ReComp` can be notified of changes in the `ClinVar` DB). `ReComp` progresses through a loop that

involves (i) quantifying the extent of the changes, (ii) estimating their impact on a cohort of past outcomes (each being the result of one execution instance and documented by its provenance) and selecting those for re-execution, (iii) a step to identify the minimal process fragments to re-execute for each of the selected instances, followed finally by requests to the deployed instances to re-execute as needed. Note that HDB is used throughout, as explained later (Sec. III-B).

The Core architecture is presented in Sec. III. Examples of process-specific elements, namely data-diff and impact estimation functions, are given in Sec. IV.

II. Related Work

To the best of out knowledge, the most recent account on state-of-the-art approaches to the general problem addressed by ReComp is the 2018 report on the first workshop on *Problems and techniques for Incremental Re-computation: provenance and beyond* [6]. Workshop contributions covered the spectrum of *Re-computation*, defined as "the repeated execution of a process, all or in parts, under slightly different inputs or configuration each time, and making use of one or more prior execution baselines as a basis for optimization".

Within this context, work that makes a connection between re-computation and process provenance is perhaps the closest to ReComp, specifically Ashish Gehani's talk on "Supporting Incremental Re-Computation with Whole System Provenance: Issues and Approaches". The approach is centred on SPADE, a framework for capturing fine-grained system-level provenance information that can later be used to improve process re-execution [7]. A related, but older approach enables *smart re-run* of Kepler workflows (SRM) [8]. The idea is to react to changes in one or more parameters in a workflow actor by only executing those parts of the workflow that are affected by the changes, taking data dependencies into account. The approach relies on coarse-grain provenance traces and intermediate data collected and stored during workflow execution, and is derived from a similar approach implemented in VisTrails [9]. Both works inspired ReComp's partial re-execution mechanism. On one hand it relies on provenance collected from workflow runs, similar to SRM, while on the other hand, to calculate the minimal re-computation subgraph we use the data versioning mechanism provided by e-SC, which is closer to file versioning in SPADE. Although the idea is not new, ours is the first implementation to operate off the e-SC workflow model, and it plays only a partial role in a more ambitious picture, where we seek to prioritise re-execution within a large collection of prior outcomes.

Slightly more peripheral to ReComp is the area of *Incremental computation*, which can be viewed as one of the ways re-computation can be optimised, however it is arguably more general, as it does not require a prior baseline because a first time executions may be incremental. Relatively older research [10], [11] has addressed the problem of reacting effectively to incremental changes in the program's input data. These techniques are based on dependency graphs, memoisation and partial evaluation – concepts similar to what we use to re-compute our process, yet applied on the scale of a single algorithm or program.

A number of incremental computation solutions has also been applied to Big Data problems. Most notable are DryadInc [12], Haloop [13] and Incoop [14] and more recently iiHadoop [15]. Again, the main difference between these and our approach is that we consider re-computation in broader sense, not limited to only a single algorithm or execution for which input data has been updated. We aim to reduce at the same time the number of past executions which need re-computation, as well as the amount of processing required within each execution. To achieve this goal, other more recent incremental techniques can in principle be considered as the basis for re-execution. These include *differential dataflows* [16] and parallel incremental computation, implemented in iThreads [17].

III. ReComp Architecture

A. External services

A high level view of the ReComp architecture is shown in Fig. 2. The Core consists of the ReComp Loop service and the History Database (HDB), and is supported by three external services, for data-diff, impact functions, and selective re-execution. Any realisation of ReComp for a specific user-defined process and dataset will need to provide an implementation for the corresponding three service interfaces, defined in List. 1. Specifically, `compute_difference` calculates the difference between two datasets that flow through a selected input, called a *process input port*, during two subsequent executions. The result is a data structure, called `difference set`, which is then used by the `compute_impact` service in conjunction with a `past_output`, to provide an estimate of the impact of the changes on *that* output. In the current implementation, impact is binary (either there is some impact, or there is none). Finally, the `reexecute_process` function is used to rerun a past process execution using a special reference structure which we call the *restart tree*. The tree, presented in detail in [18], describes how different parts of both simple and complex hierarchical processes have been affected by given data changes.

We follow a common dataflow abstraction for computation, where each computation component has input and output ports, and ports are connected through data channels to form a computation graph. Following this abstraction, input and output data are bound to input/output ports during execution. As there are potentially multiple inputs and outputs, including intermediate outputs from some of the components, multiple `compute_difference` and `compute_impact` services can be deployed, each associated with a different input and output port, respectively (but

Listing 1. The interface of the three external functions used by the ReComp Loop service.

```
function compute_difference(old_data, new_data,
    ↪ config) : difference_sets

function compute_impact(past_output,
    ↪ difference_sets : array, context_data :
    ↪ array, config) : {0, 1}

function reexecute_process(restart_tree, config)
    ↪ : execution_id
```

not all ports must have an impact functions associated with them). The ReComp Core will have access to the bindings of data to ports through the provenance trace. It is, however, the responsibility of the process runtime environment to capture that information.

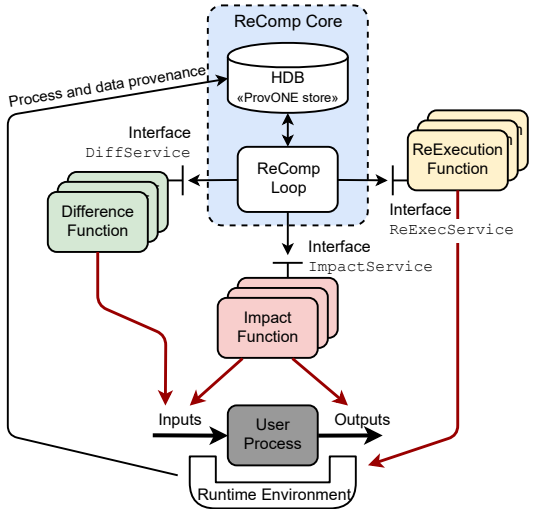

Figure 2. The two core components of the ReComp framework, the *ReComp Loop* service and *History DataBase*, control the re-computation of a user-defined process via a set of external functions.

B. The History Database (HDB)

HDB is a provenance store and query service used to manage coarse-grain provenance facts about data and processes. To store the facts we use the ProvONE data model [19], an extension of the generic PROV standard [20], that enables us to capture information about simple black-box processes as well as more complex workflow structures. Additionally, we use HDB to maintain ReComp-specific process annotations such as references to external services and their configuration parameters. Key elements of the model are the `document` and `execution` statements, which are equivalent to `entity` and `activity` in PROV. These are used to describe information about data artifacts and processes involved in generating the artifacts, the so called *retrospective provenance*.

Listing 2. An example of three requests to retrieve, store and update simple provenance facts in HDB.

```
GET /hdb/document?id=doc-0

PUT /hdb/execution?id=ex-1&startTime=2019-05-13

POST /hdb/used?id=usg-2&opType=update
{
   "prov:activity: "ex-1",
   "prov:entity": "doc-0"
}
```

Listing 3. An example of a HDB query to retrieve port information related to a specific usage statement.

```
POST /hdb/query
{
   "query": "used(_UId, 'ex-1', 'doc-0', _, _),
       ↪ hadInPort(_UId, PId), port(PId, PAttrs)"
}
```

What distinguishes ProvONE from PROV, however, is its ability to capture also *prospective provenance*, i.e. the workflow structure of the process that generates a data artifact. This is in contrast to *retrospective provenance*, which contains a trace of a specific past process execution and how it consumed particular input data. ProvONE, instead, includes `program` and `port` statements which may be used to define the structure of the user process, its inputs and output ports. We use these statements to instrument the process so the external functions can be attached appropriately.

To store and retrieve provenance information, HDB exposes a dedicated two-level API. The basic interface operates on simple provenance facts (List. 2), whereas the more generic interface can execute user-defined queries (List. 3). The queries are expressed using Prolog, the underlying implementation language of HDB. Prolog offers great flexibility in constructing queries and also enables stored rules to be invoked. For example, query in List. 3 retrieves information about all ports through which execution `ex-1` used document `doc-0`. That information is required to obtain the details of a difference service attached to process ports. We discuss the interaction between HDB and other components of the framework in the following section.

C. The ReComp Loop Service

The ReComp Loop service coordinates the overall re-computation of processes. It reacts to the change events, uses difference and impact services to analyse the impact of the changes on past executions and submits a subset of affected executions to rerun.

The sequence diagram in Fig. 3 shows how the Loop service reaches a recomp/no-recomp decision for a given data change and then triggers re-execution, by interacting with the external services. The change notification, in the

form of a PROV data derivation statement, is stored in the HDB and triggers the Loop service. The notions of *recomputation front* and *restart trees* are presented in detail in [18] and are not further discussed here. Suffice it to say that each restart tree is a composite process reference considered as an atomic unit of possible re-computation.

In the loop, each restart tree is considered in turn. For each tree all referenced `compute_difference` services are invoked (cf. "for all annotated ports"). Then for each *non-empty* data difference set, all the available `compute_impact` functions found downstream of the related port are invoked. If all the functions detect no impact, that port is removed from the tree and the tree is pruned. The overall result of this impact analysis run for each restart tree is a decision whether or not the referenced process should be recomputed. If after pruning the tree remains non-empty, the decision is positive and the execution service is activated.

IV. Customizing the ReComp Framework

In order to use the `ReComp` framework to control the recomputation of a user-defined process a few steps need to be taken into account initally (Fig. 4). Of these the two most important are to understand input data changes and the impact they may have on process outputs. Once this is achieved, the external functions to compute differences, impact and re-execute the process are needed. Finally, the process needs to be annotated to link the functions to specific fragments of the process, so `ReComp` can take control over its re-computation.

A. Understanding Changes and Impact

Although changes in data and their impact on past outputs are, usually, very specific to each user-defined process, a common aspect behind re-computation of any process is the frequency of data changes and how deep impact they have on process outputs. For example, if data changes are frequent and only a small subset of them is likely to affect outputs, there is an opportunity to optimise re-computation. Conversely, if every change invalidates previous result completely, `ReComp` can only be used to automate re-computation but will not be able to reduce its cost.

To illustrate these points, in Fig. 5 we show the impact of changes of eight consecutive versions of the ClinVar database ($08/15$–$03/16$) on the outputs of the SVI workflow for three patient samples classified using ClinVar version $07/15$. The black square denotes a significant change in sample classification, whereas dots show insignificant changes or no change at all. In this example, taken from our larger study [4], a *significant* change is one that causes the patient's diagnosis to change, i.e., either because a new pathogenic variant has been discovered, or because a patient's variant previously thought to be pathogenic has now been found to be benign. Other, more subtle measures of impact are possible, too. Importantly, however, in these three cases and given a function which can accurately estimate the impact as defined above, `ReComp` may reduce the number of SVI re-executions from 24 to only 5.

However simple this exploratory analysis, it becomes a clear and strong motivation to follow the proposed approach and implement the `ReComp` external functions. Later in this paper we briefly outline a few examples of a difference and impact functions. For now, however, we focus on the final step needed to bring a process under `ReComp` control – the annotations.

B. Process Annotations

For `ReComp` to be able to control re-computation we have to instrument the process of interest, to let our framework know how data difference and impact may be computed and how the process may be rerun. List. 4 shows an example of a hypothetical program with one input and one output port, annotated as needed by `ReComp`.

The first statement in the example specifies program `prog-0` and identifies a service that can rerun it (lines 1–4). The three `ReComp`-specific annotations refer to (1) the interface implemented by the given service, (2) the location where the service is available, and (3) optional configuration that is passed when making a service call, respectively. The service implements interface `uk.org.recomp.ReExecService` that declares function `reexecute_process` presented earlier in List. 1.

Then, the listing describes two ports. Input ports, like `input-0` in lines 6–10, are annotated in a way very similar to programs. They link a port with a service that implements the `compute_difference` function, such as the `uk.org.recomp.DiffService` interface. Annotation of output ports is slightly extended, however, because impact functions need more details to run. Lines 12–16 are equivalent to the previous program and input port annotations. Line 17 refers to a list of input ports which the impact function needs a difference set of (port `input-0` in this example). List 18 refers to the list of context ports, i.e. input ports which data is passed to the function as-is, without computing difference. In the SVI use case this may be, for example, a patient phenotype hypothesis which determines a subset of genes an impact function needs to consider.

The annotations declare also configuration parameters defined in documents `*-config-0`. These are function-specific parameters, like an impact threshold value in the flood modelling use case, which the Loop service passes on to the function every time a call is made.

C. Binding between Program, Port and Past Executions

As shown above, the effort to annotate a process, such that `ReComp` is able to control its re-computation, is rather minimal. We only need to declare which ports have a difference or impact function attached to them and add

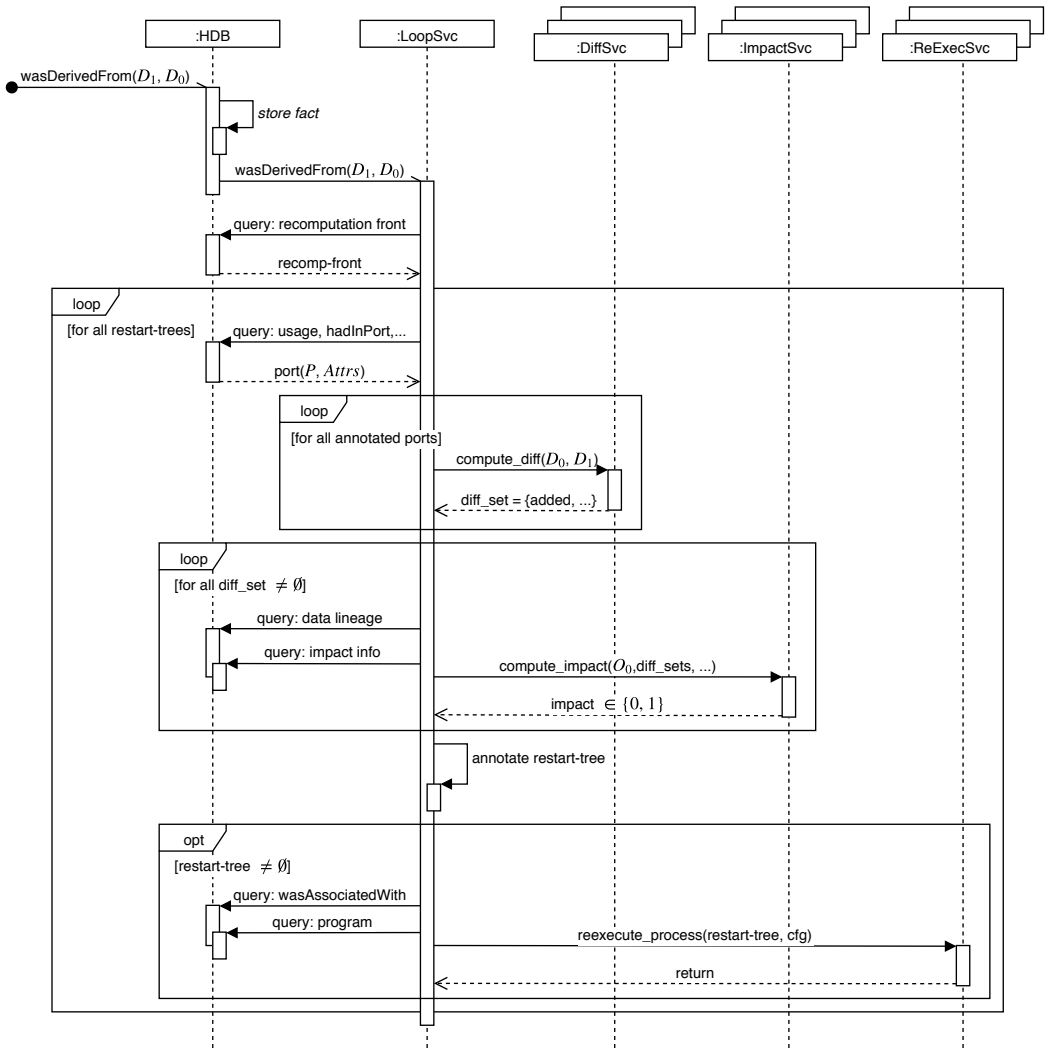

Figure 3. Key interactions between the ReComp Loop service and other framework components.

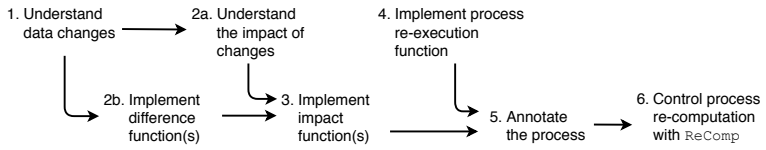

Figure 4. Common steps to harness a user-defined process with the ReComp framework.

Patient Id	ClinVar version 08/15	09/15	10/15	11/15	12/15	01/16	02/16	03/16
C_0051	■	·	·	·	·	·	■	·
C_0065	■	·	·	·	·	·	·	■
D_1136	·	·	■	·	·	·	·	·

Figure 5. Impact of changes in the ClinVar database on the output of the SVI workflow.

a statement with a reference to a service that is able to re-execute the process.

As mentioned earlier, however, we assume that the process runtime environment is able to accurately capture *retrospective provenance*, i.e. the steps involved in the generation of past process outputs. ReComp needs retrospective provenance to be able to bind past process executions with relevant ports and programs, and so be able to find the annotations.

This binding is possible only if the runtime environment is able to capture port related details during process execution. Fig. 6 shows a relevant part of the ProvONE conceptual model which links together the retrospective and prospective representation of the user process. Whilst this requirement is not very stringent and many environments, especially workflow management systems, can fulfil it, our framework will only be able to control the re-computation of a given process if the above information is captured using the ProvONE statements. That may need an implementation of

Listing 4. Annotations attached to a process and its two ports using the ProvONE data model and ReComp specific attributes.

```
1  program(prog-0, [
2    recomp:reexec-interface =
         ↪ "uk.org.recomp.ReExecService",
3    recomp:reexec-svc-url = "http://...",
4    recomp:reexec-config = "reexec-config-0" ]).

6  port(input-0, [
7    prov:label = "An input port",
8    recomp:diff-interface =
         ↪ "uk.org.recomp.DiffService",
9    recomp:diff-svc-url = "http://...",
10   recomp:diff-config = "diff-config-0" ]).

12 port(output-0, [
13   prov:label = "An output port",
14   recomp:impact-interface =
         ↪ "uk.org.recomp.ImpactService",
15   recomp:impact-svc-url = "http://...",
16   recomp:impact-config = "impact-config-0"],
17   recomp:impact-input-ports = [ "input-0" ],
18   recomp:impact-context-ports = []).

20 hasInPort(prog-0, input-0).
21 hasOutPort(prog-0, output-0).

23 document(diff-config-0, [ ... ]).
24 document(impact-config-0, [ ... ]).
25 document(reexec-config-0, [ ... ]).
```

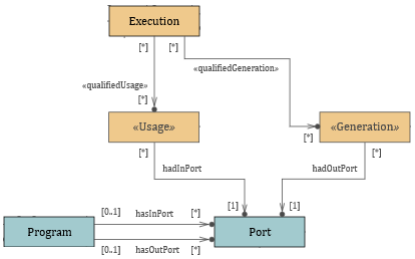

Figure 6. A part of the ProvONE conceptual model relevant to ReComp, which binds the retrospective and prospective provenance representation.

a provenance wrapper or adapter component as discussed later.

V. Applications to the Selected Case Studies

In this paper we focus on two very distinct processes: a compute intensive flood modelling simulation tool (black-box) and a data analysis workflow that helps classify pathogenicity of patient genetic variations (grey-box).

A. Black-box Process – Flood Modelling

CityCAT is a home-grown tool developed at Newcastle University, which takes a number of user inputs including the digital elevation model (DEM), set of building and green area polygons, configuration of the rainfall event

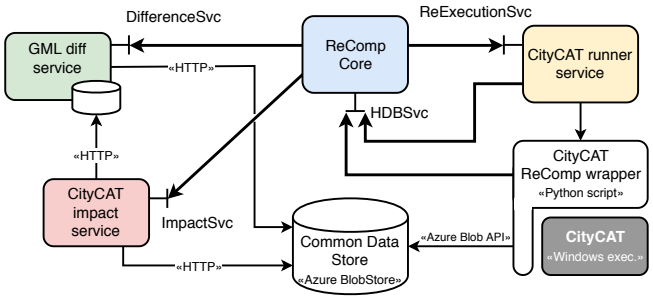

Figure 7. ReComp components supporting the re-computation of flood modelling tool *CityCAT*.

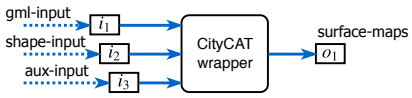

Figure 8. The CityCAT program definition using the ProvONE data model.

and some additional configuration parameters, and computes a sequence of surface maps with the predicted level of water at subsequent time points. Although ReComp could potentially be used to monitor all these inputs, together with the evolution of the CityCAT software itself, we limit our study to changes in the city area and infrastructure, and their impact on the output surface maps. The updates to the infrastructure are provided by the DigiMap Ordnance Survey in the UK every six months.[3]

The tool is a single Windows executable, thus two simple adaptations are required to bring it under ReComp control. Firstly, we need to adapt the process because CityCAT reads and stores data in the local filesystem, thus to let it interact with ReComp deployed on the Cloud we need to make the input and output data available to the ReComp Core and the external functions. Secondly, the executable is a simple black-box process that does not generate data and process provenance information. Thus, we implemented a wrapper script that can download inputs and upload simulation outputs to a common data store (Azure BlobStore in our case) and push the corresponding provenance facts into HDB (Fig. 7). The wrapper script defines how the CityCAT process and its executions are represented in provenance traces. As depicted in Fig. 8, we defined the process with three input and one output port. The `gml-input` and `shp-input` are equivalent and allow the input polygon data to be passed in two different formats GML and SHP, respectively. The `aux-input` aggregates all other input files which CityCAT requires to run, such as the elevation model and rainfall event configuration. For the purpose of our experiments we consider these files as being static.

Difference and Impact Functions: We developed a generic difference service that takes from two versions

[3]https://digimap.edina.ac.uk/os

Listing 5. Some of the annotations linked to two CityCAT ports.

```
port(gml-input, [
  prov:label = "GML input",
  recomp:diff-svc-url = "http://...", ... ])

port(surface-maps, [
  prov:label = "Surface maps",
  recomp:impact-input-ports = ["gml-input"],
  recomp:impact-config = "impact-config-0", ... ])

document(impact-config-0, [
  threshold = 0.2, min-depth = 0.2,
  dilation = 15, ... ])

program(ccw-runner, [
  recomp:reexec-svc-url = "http://..." ]).
```

of building and green area polygons in the GML format and produces four change sets representing addition and removal of buildings and land. The impact function maps these differences into six change types specific to CityCAT, such as buildings that changed into land and hard surface, and analyses average water depth over the footprint of the changes. The impact function exposes three configuration parameters that allow the user to control its sensitivity. The `threshold` parameter refers to the depth of water in the footprint such that the higher the threshold, the deeper the water level that is considered insignificant. Parameters `dilation` and `min-depth` indicate the breadth of the footprint from the original change and the minimal depth of water during impact calculation, respectively. Together they control whether the function is sensitive to deep changes of small footprint or shallow but more widespread changes.

Re-execution Function: the next element required to automate the re-computation is the re-execution function. In the case of a black-box process and CityCAT specifically, this is a relatively straightforward task. The key detail here is that the function needs access to HDB and the common data store in order to retrieve past configuration and input data used to generate previous output. These details are used to configure a new re-execution of CityCAT in which only the GML input is modified.

Process Annotations: with the difference and impact functions ready to use, the final preparation step is to annotate the process. List. 5 shows the most relevant parts of CityCAT annotations, which otherwise are very similar to the ones shown previously in List. 4. We annotate only the two ports that are relevant for `ReComp`, i.e. `gml-input` and `surface-maps`. Additionally, we pass configuration of the impact function to configure function's sensitivity. The last two lines refer to the re-execution service which is able to rerun CityCAT via the wrapper script.

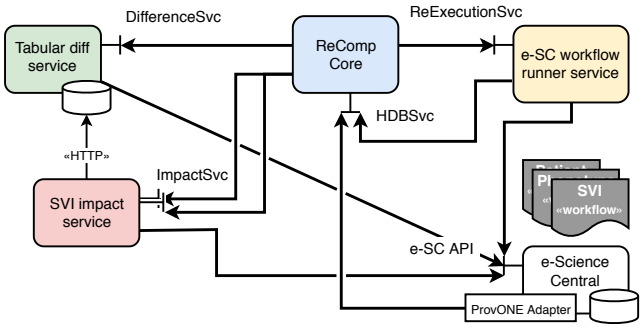

Figure 9. `ReComp` components supporting the re-computation of the SVI workflow.

B. Grey-box Workflow – Simple Variant Interpretation

SVI is a workflow that takes annotated patient variants and suspected phenotype and classifies the variants with a simple traffic light system of the red, green and amber colour to denote pathogenic, benign and variants of unknown or uncertain pathogenicity, respectively. It relies on two external databases: one, like OMIM GeneMap, helps to translate phenotype description into a broad set of related genes; the other, e.g. NCBI ClinVar, is used to determine whether genetic variation may have deleterious effect on genes function.

The overall approach to harness SVI with `ReComp` is similar to the black-box case presented earlier. We have to implement the three external functions and annotate process ports. In detail, however, there is a number of differences in how this is achieved (Fig. 9).

Process Adaptations: SVI as a workflow runs on e-Science Central. The system includes its own data storage, provenance capture facility and API to control and manage workflows. Thus, there is no need to implement a wrapper. Instead, `ReComp` external functions can use the e-SC API to access input and output data from the WfMS directly. Regarding the provenance information, however, e-SC captures them using a customised OPM model [21]. Therefore, we implemented a simple adapter which can translate provenance statements from the e-SC specific model into the generic ProvONE model. Using the adapter e-SC is able to communicate the provenance information directly to the `ReComp` core via the HDB interface.

Difference and Impact Functions: as in the previous example we implemented a generic difference function which can compute changes between any two tabular data sets and produces four subsets of added, removed and two versions of changed records. The function can compute a generic more detailed difference, or more precise difference that indicates only changes relevant to the given process (for details see [4]). The way it operates is configured via port annotations.

The difference sets are then used by impact functions

to estimate their influence on past process outcomes. With SVI being a workflow we have partial insight into the structure and semantics of its parts. Thus, we implemented multiple, simpler impact functions overlooking the influence of changes on different parts of the process. For example, one of the functions measures the impact of changes in OMIM GeneMap on the set of genes relevant to the user-defined phenotype hypothesis. Another one takes into account changes of pathogenicity in ClinVar and computes impact on the final SVI output. And to simplify the implementation we included all the functions into a single *SVI impact service*.

Re-Execution Function: to rerun SVI which is managed by e-Science Central we used system's dedicated workflow management API. As previously, the key part of re-execution is to replicate the exact configuration of a past execution and amend only the relevant parts indicated by the restart tree. But given the knowledge about the structure of the workflow enables also more sophisticated re-execution in which only parts of a workflow are rerun.

The idea, similar to the smart rerun of scientific workflows [22], [23], can reduce the cost of re-execution. In case of e-SC workflows, however, it requires additional effort to edit workflow structure programmatically because e-Science Central does not support partial workflow re-execution. We left the implementation of this aspect for the future. Our re-execution function can simply rerun the complete workflow with configuration based on a selected previous execution and updated as required.

Process Annotations: except for the number and selection of ports and programs, annotating a workflow is very similar to doing so for a black-box process. Being able to rely on the runtime environment to capture statements on the verge between the retrospective and prospective provenance, i.e. `hadInPort` and `hadOutPort`, makes annotating a user process as simple as it could be. We only need to make sure that port and process identifiers of annotated ports match those used by the WfMS.

VI. Benefits and Costs – Lessons Learnt

Here we discuss the effort required to bring a new process under `ReComp` control, and we compare it with the full re-computation cost when `ReComp` is not used.

The two use cases presented in this paper are intentionally very different in nature, and so allow us to look at the problem of process re-computation from very different angles. CityCAT is a compute intensive black-box tool which implements a complex hydraulic model and can take hours to run on a multi-core system, whereas SVI is a simple data analytics workflow that runs in the range of minutes usually over a large cohort of patients.

Based on our experiences from adapting `ReComp` to the two selected use cases, in Tab. I we present more details of how the differences between them affect the effort required to bring them under `ReComp` control and what optimisation opportunities arise from it.

Table I
Key aspects learnt from the adaptation of the ReComp framework to the two selected use cases.

Aspect	Use Case: Flood Modelling	Use Case: SVI
Process Type	black-box → easy to manage	grey-box (workflow) → medium–hard to manage
Experiment Structure	one–a small number of runs	a set of phenotypes, each with a cohort of patients
Change(s)	single changing input; low rate (biannual)	two changing inputs; medium/high rate (monthly/daily)
Process Adaptation	*medium* effort (wrapper + provenance handler)	*medium* effort (provenance adapter)
Difference Function(s)	low–*medium* effort (some expert knowledge needed)	low effort (simple tabular difference)
Impact Function(s)	single anchor point; **very hard** to implement	multiple anchor points; low–*medium* effort
Re-execution Function	low effort	*medium* effort (some expert knowledge needed)
Optimisation Opportunities	low: opaque structure	high: partial re-execution possible
	medium: 30–60% reruns unneeded	high: typically >90% reruns unneeded

CityCAT being a black-box process is relatively straightforward to manage, nonetheless, it requires some effort to develop a wrapper script. The script's main job is to expose data used by the tool and to capture provenance information as required by the framework. SVI as a workflow requires more sophisticated approach to re-computation. Despite being made up of a relatively simple graph of tasks, in order to handle full-fledged workflows `ReComp` must consider more sophisticated structures, including recursively nested sub-workflows. To do so it uses the notion of restart tree mentioned earlier and defined in [18]. Additionally, to control SVI we had to develop a provenance adapter which can translate provenance facts captured by e-SC, the WfMS it runs on, into the ProvONE format needed by `ReComp`.

The next aspect is the structure of a single experiment which also differs considerably between Flood Modelling and SVI. CityCAT simulations are usually run once for a single or a small number configuration settings to model e.g. different rainfall events. SVI, however, is typically used to analyse cohorts of patients spread across a set of different phenotypes, while all these analyses use the same reference databases. That, together with the processing runtime and the rate of changes of process inputs, determine opportunities for

optimisation that arise from re-computation.

The rate of changes for For CityCAT is relatively low as updates to urban infrastructure are published about twice a year via the DigiMap web service. However, the computation complexity of the tool and the manual effort of a domain expert needed to analyse the results set the cost of re-computation very high. In contrast, in the case of SVI input data may change rapidly. The OMIM GeneMap database is updated daily, while new version of NCBI ClinVar is published every month. Thus, despite the rerun of a single SVI execution is relatively cheap, multiplied by the number of changes and analysed patient samples set the total cost of re-computation to a high level, too.

Next, we compare the re-computation cost with the effort required to bring the two processes under `ReComp` control. Firstly, however, we note that the development effort is made only once, and so will be offset by running `ReComp` for longer period. Secondly, our judgment of effort is relative and intentionally imprecise as different development teams with different skills would solve the problem at different speeds.

From the experience we gained the easiest tasks were to develop the re-execution function for the CityCAT wrapper. It involved the implementation of the `ReExecService` interface and starting a wrapper process. As CityCAT is a black-box, the amount of reconfiguration work needed to instantiate a new process based on the past configuration was relatively small. Implementing the function for SVI required more work because SVI included a number of tasks which differ from patient to patient and phenotype to phenotype. It also required some expert knowledge to reconfigure existing e-Science Central workflow using provenance data from past workflow invocations.

Another low effort task was to develop a difference function for tabular data. This was implemented using Python Pandas library[4] which offers useful constructs to build it. However, other implementations could use a SQL engine equally easy. A difference function for the CityCAT input data in the SHP and GML formats was also relatively straightforward to code but it required some expert knowledge on the use of geospatial libraries and understanding of the specialized file formats.

Of medium effort was also the task of the initial adaptation of processes. In the case of Flood Modelling the key part was to design the wrapper. Due to lack of ready to use ProvONE python libraries, the main effort went into the design and implementation of a simple provenance capture mechanism that stores information in HDB in the ProvONE data format. Similar, although slightly more effort was needed to code a provenance adapter for e-SC. If conversion from the dedicated OPM model to ProvONE was relatively easy, a little more work and expert knowledge was needed to capture details on the verge between prospective and retrospective representation (the `hadInPort` and `hadOutPort` statements). They have not been captured by e-SC previously.

Ultimately, most of our effort was spent on the implementation of the impact functions. As SVI is a relatively simple tool, the impact analysis, and then design and implementation of appropriate functions was not too difficult. Additionally, given the workflow structure we could code multiple simpler functions rather than one more sophisticated. But in the case of Flood Modelling scenario, the design and development of the impact function was a single most difficult task. It required good understanding of the input changes and the outputs. It also required expert discussion on potential approach and extensive trial-and-error effort. And due to the nature of the problem, finding the impact of change still remains a challenge in specific corner-cases for which only running the full hydraulic model could give the accurate answer.

A quantitative evaluation of the benefits that `ReComp` brings to data analytics is necessarily going to be on a case-by-case basis, and is outside the scope of this paper. We are currently testing the `ReComp` concept on additional case studies. The healthcare space provides particularly good examples that call for efficient solutions to the recurring re-computation of computationally expensive analytics processes, where rapidly evolving domain knowledge combine with the dynamic nature of big datasets (for instance, patients' Health Records).

[4]http://pandas.pydata.org/

ACKNOWLEDGMENT

This work has been supported by EPSRC in the UK [grant no.: EP/N01426X/1] and a grant from the Microsoft Azure for Research programme.

REFERENCES

[1] J. Cala, E. Marei, Y. Yu, K. Takeda, P. Missier, J. Cała, E. Marei, Y. Xu, K. Takeda, and P. Missier, "Scalable and Efficient Whole-exome Data Processing Using Workflows on the Cloud," *Future Generation Computer Systems, Special Issue: Big Data in the Cloud*, vol. 65, no. Special Issue: Big Data in the Cloud, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.future.2016.01.001

[2] R. Bertsch, V. Glenis, and C. Kilsby, "Urban flood simulation using synthetic storm drain networks," *Water (Switzerland)*, vol. 9, no. 12, 2017.

[3] P. Missier, E. Wijaya, R. Kirby, and M. Keogh, "SVI: A Simple Single-Nucleotide Human Variant Interpretation Tool for Clinical Use," in *Data Integration in the Life Sciences*, N. Ashish and J.-L. Ambite, Eds. Springer International Publishing, 2015, pp. 180–194.

[4] J. Cała and P. Missier, "Selective and Recurring Re-computation of Big Data Analytics Tasks: Insights from a Genomics Case Study," *Big Data Research*, vol. 13, pp. 76–94, sep 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2214579617303520

[5] H. Hiden, S. Woodman, P. Watson, and J. Cala, "Developing cloud applications using the e-Science Central platform," *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, vol. 371, no. 1983, jan 2013.

[6] P. Missier, T. Malik, and J. Cala, "Report on the First International Workshop on Incremental Re-computation: Provenance and Beyond," {*SIGMOD*} *Record*, vol. 47, no. 4, 2018. [Online]. Available: https://sigmodrecord.org/2019/05/02/report-on-the-first-international-workshop-on-incremental-re-computation-provenance-and-beyond

[7] H. Lakhani, R. Tahir, A. Aqil, F. Zaffar, D. Tariq, and A. Gehani, "Optimized Rollback and Re-computation," in *2013 46th Hawaii International Conference on System Sciences*, no. i. IEEE, jan 2013, pp. 4930–4937.

[8] I. Altintas, O. Barney, and E. Jaeger-frank, "Provenance Collection Support in the Kepler Scientific Workflow System," *Work*, vol. 4145, pp. 118–132, 2006.

[9] L. Bavoil, S. Callahan, P. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo, "VisTrails: Enabling Interactive Multiple-View Visualizations," in *VIS 05. IEEE Visualization, 2005.*, no. Dx. IEEE, 2005, pp. 135–142.

[10] U. A. Acar, G. E. Blelloch, M. Blume, R. Harper, and K. Tangwongsan, "An experimental analysis of self-adjusting computation," *ACM Transactions on Programming Languages and Systems*, vol. 32, no. 1, pp. 1–53, 2009.

[11] G. Ramalingam and T. Reps, "A categorized bibliography on incremental computation," *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages – POPL '93*, pp. 502–510, 1993.

[12] L. Popa, M. Budiu, Y. Yu, and M. Isard, "DryadInc: Reusing work in large-scale computations," *HotCloud'09 Workshop on Hot Topics in Cloud Computing*, pp. 2–6, 2009. [Online]. Available: http://static.usenix.org/events/hotcloud09/tech/full_papers/popa.pdf

[13] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "HaLoop," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, sep 2010.

[14] P. Bhatotia, A. Wieder, R. Rodrigues, U. a. Acar, and R. Pasquin, "Incoop: MapReduce for incremental computations," *Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC '11*, pp. 1–14, 2011.

[15] A. G. Bin Saadon and H. M. O. Mokhtar, "iiHadoop: an asynchronous distributed framework for incremental iterative computations," *Journal of Big Data*, vol. 4, no. 1, p. 24, dec 2017.

[16] F. D. McSherry, D. G. Murray, R. Isaacs, and M. Isard, "Differential Dataflow," in *6th Biennial Conference on Innovative Data Systems Research (CIDR '13)*, 2013. [Online]. Available: http://cidrdb.org/cidr2013/Papers/CIDR13_Paper111.pdf

[17] P. Bhatotia, P. Fonseca, U. A. Acar, B. B. Brandenburg, and R. Rodrigues, "iThreads," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 645–659, 2015.

[18] J. Cała and P. Missier, "Provenance Annotation and Analysis to Support Process Re-computation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 11017 LNCS, pp. 3–15. [Online]. Available: http://link.springer.com/10.1007/978-3-319-98379-0{_}1

[19] V. Cuevas-Vicenttín, B. Ludäscher, P. Missier, K. Belhajjame, F. Chirigati, Y. Wei, S. Dey, P. Kianmajd, D. Koop, S. Bowers, I. Altintas, C. Jones, M. B. Jones, L. Walker, P. Slaughter, B. Leinfelder, and Y. Cao, "ProvONE: A PROV Extension Data Model for Scientific Workflow Provenance," 2016. [Online]. Available: http://jenkins-1.dataone.org/jenkins/view/DocumentationProjects/job/ProvONE-Documentation-trunk/ws/provenance/ProvONE/v1/provone.html

[20] L. Moreau, P. Missier, K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes, "PROV-DM: The PROV Data Model," World Wide Web Consortium, Tech. Rep., 2012. [Online]. Available: http://www.w3.org/TR/prov-dm/

[21] S. Woodman, H. Hiden, P. Watson, and P. 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*. New York, New York, USA: ACM Press, 2011, pp. 127–136. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2110497.2110512

[22] I. Altintas, O. Barney, and E. Jaeger-frank, "Provenance Collection Support in the Kepler Scientific Workflow System," *Work*, vol. 4145, pp. 118–132, 2006.

[23] H. Lakhani, R. Tahir, A. Aqil, F. Zaffar, D. Tariq, and A. Gehani, "Optimized Rollback and Re-computation," in *2013 46th Hawaii International Conference on System Sciences*, no. i. IEEE, jan 2013, pp. 4930–4937.