INTELLIGENT CLIENT WORKFLOW SYSTEM FOR THE MODERN WEB

by

PETER RYAN ELESPURU

B.S., University of Colorado, 2002

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirement for the degree of

Master of Science

Department of Computer Science

2013

This thesis entitled:

Intelligent Client Workflow System for the Modern Web

written by Peter R. Elespuru

has been approved for the Department of Computer Science

_____

Prof. Shivakant Mishra

_____

Prof. Willem Schreuder

_____

Dr. Eric Kihn

Date_____

The final copy of this thesis has been examined by the signatories, and we Find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Elespuru, Peter Ryan (M.S., Computer Science)

Intelligent Client Workflow System for the Modern Web

Thesis directed by Associate Professor Shivakant Mishra

ABSTRACT

The modern web browser is becoming ever increasingly more powerful in many ways; this work will look at it very specifically in terms of computing resource potential, as a general computation workflow system, with a server side coordination component hosted in the cloud. The unique contribution of this work is its utilization of client computing resources via the web browser, with minimal cloud based coordination, wherein the end results are an extremely capable and intelligent client side system. Its reliance on the web browser also means it relies heavily on current and forthcoming web standards. Web standards continue to evolve and include both specific and general features with an eye toward computation, not just simply rendering web content any longer. This paper and its associated project work seek to approach those problems from a specific angle, namely, a web based client workflow system, wherein the user interface, execution engine, and advanced capabilities all combine and maximize the client side potential. The unique combination of visualization, creation and computation in the web browser client, with coordination through the cloud, yields a new and different workflow and computation system that is well suited as an exploratory scientific analysis platform, called FlowCloud.

DEDICATION

Dedicated to everyone who helped or supported me along the way…

ACKNOWLEDGEMENT

I want to thank my advisor, Prof. Shivakant Mishra for his support, patience and guidance throughout this work. His thorough knowledge and immense expertise in the distributed systems domain were invaluable throughout the entire course of this project.

I would also like to thank my co-advisers, Prof. Willem Schreuder and Dr. Eric Kihn. Professor Schreuder's expertise in the graphics and GPU computation domain were the reason these aspects of the work were possible. Dr. Kihn's vast scientific knowledge and familiarity with all things software helped provide the context for this work, without which it would have been aimless.

CONTENTS

CHAPTER

TABLES

Table

FIGURES

Figure

CHAPTER I

INTRODUCTION

A frequently identified challenge facing the software world is that of doing something meaningful with not only the ever increasingly large volumes of data, but more importantly, combining data from different sources and quickly and easily manipulating it in various ways. This is an area which is immensely useful to the science community specifically, and which is currently extremely under served by available tools. Most science domains have a decent handle on storing their data, making it accessible to the public as well as other scientists, and properly describing it with metadata for posterity. The problem is that there are so many different data formats and standards, that each is effectively in its own silo, and even simple comparative analyses are difficult, if at all possible. An important question then is the following: can a system be created which allows a scientist to easily obtain data from one or more sources, perform basic analyses on it, and visualize the result, all without traditional data center server resources, and with a simple to use and understand interface? Computer hardware infrastructure is costly and potentially difficult to obtain, and current software systems are both large and complex. The result is that getting up and running simply to perform some basic scientific analysis requires too much knowledge of unrelated domains for scientists in the authors' opinion. Creating an intelligent client side system could provide a fast and easy way to root out much more valuable research in a wide range of data sets. The

results of this thesis research do just that, they provide a graphical user interface for building a workflow, which can obtain data from multiple sources, perform basic analyses, and visualize the results, all entirely within the web browser. The workflow execution even happens in the user's web browser, while data and content are stored in the cloud and hosted by their original sources. The unique take on this problem is the movement of the workflow execution into the web client, creating a smart client containing creation, visualization and execution, lacking only the data itself in the client. This is significant because at this time, all other workflow systems capable of consuming and processing these science data require traditional low level infrastructure model data center resources, and sizable server side components in most cases. Moving flow creation, visualization and execution to the client, and data and content to the cloud, obviates the need for such data center resources altogether in most cases. The need to have either an actual typical infrastructure, or access to an infrastructure as a service cloud makes the barrier to entry higher than necessary for the sake of setting up and using one of the existing workflow systems. Lowering this barrier, along with the one that exists for overall usability in terms of the analysis tools, and incorporating new and unique aspects of modern web platforms, are what set this work apart from other efforts in place or in development. The nature and scale of the intelligence within the client, relative to the targeted domain and its existing tools, is what makes this new and unique, and the implementation that stemmed from this work has been named FlowCloud. Underpinned and interconnected by web data services, this project encapsulates workflow creation, execution and visualization into the intelligent client.

CHAPTER II

BACKGROUND TECHNOLOGIES

2.1 Web Services

The topic of web services is very large and too general, so to begin with a quick summary of what this means in the context of this project is in order, to provide the background needed to understand where the project fits in the grand scheme of things. Generally though, web services are an abstraction, or interface.

There are several dominant web service infrastructures in active use today, and the two most widely used are REST (representational state transfer) and SOAP (simple object access protocol) [1]. They have different strengths and weaknesses, and due to the fact that SOAP based architectures are on the decline, and REST based ones are increasing as show in figure 1, this project's implementation will focus on a RESTful approach. Note the trends in figure 1 below as well; which clearly show the rise of RESTful architectures relative to other commonly used solutions, rather dramatically in fact.

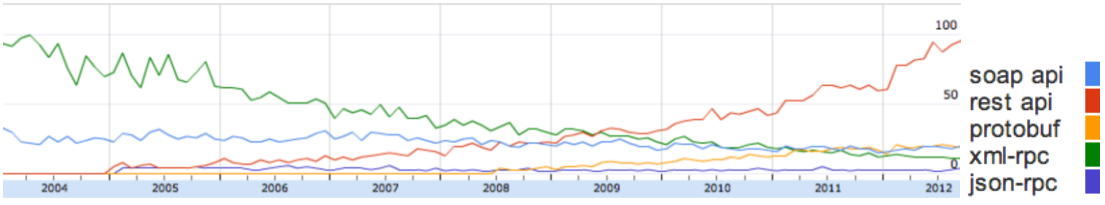

Figure 1: SOAP on the decline, REST on the rise, per Google trends

Removing RESTful API for a closer comparison of the remaining technologies additionally shows the decline of SOAP relative to other commonly used architectures, as seen in figure 2 below.

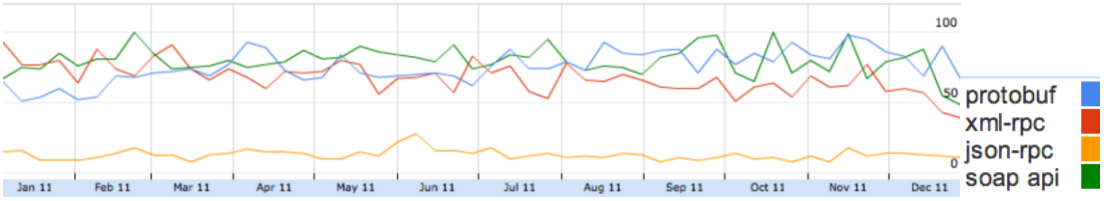

Figure 2: SOAP usage stagnant, relatively, per Google trends

2.2 RESTful Operation Primer

REST conceptually builds on top of HTTP, but quickly here are the methods and what they are responsible and used for in practice. This is not a comprehensive list of HTTP methods, just those most widely used in RESTful systems.

Operation	***Purpose***
OPTIONS	Obtain available communication options
GET	Retrieve a specified resource
HEAD	Same as get, without a content body
DELETE	Remove a specified resource
POST	Create a specified new resource
PUT	Update a specified resource

Table 1: HTTP methods used by RESTful services

2.3 RESTful Architecture High Level

This all starts to fit together by looking at the architecture, and where each piece of the larger picture lives. The choice of the underlying service architecture is paramount to the success of web based applications [2], and can be seen from a high level in figure 3 below.

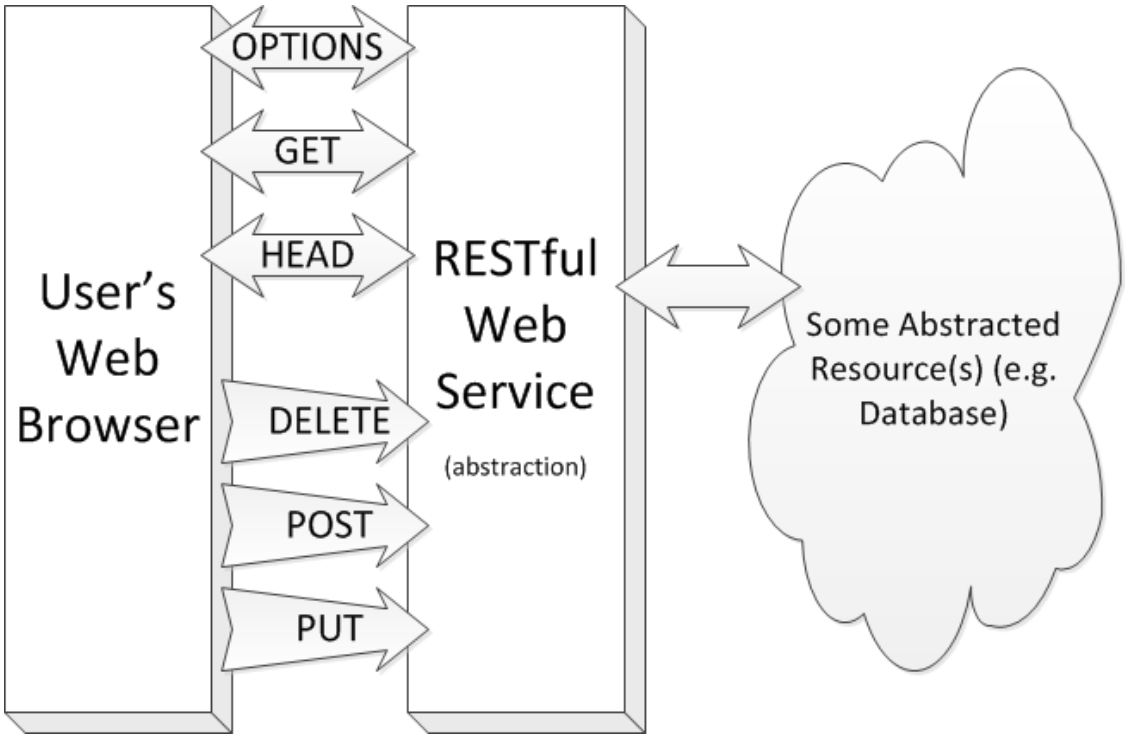

Figure 3: RESTful web services over HTTP from a high level

For the sake of this project, it is safe to think of the web services tier generally as a way for end users and other client software, as well as the project itself, to communicate and execute remote procedure calls as a way of providing an abstract interface to arbitrary underlying resources [3]. For example, as seen in

figure 3 above, a database is one such abstracted resource typically hidden by a web service interface. In the larger context, a RESTful architecture allows for easy integration by external entities, into a wide array of situations and solutions [4] and is used in most domains for the architecture of their web services today. Figure 4 below quickly illustrates where this project's work lives, in the user's web browser, coordinating with one or more RESTful web services to achieve an end result. Depending on the mechanism(s) used to interact with the web services, a proxy running locally co-located with the web application may be necessary to interact with external resources to do browser disallowing cross-domain requests for security purposes. There are a number of ways to address this in practice, and it is left to the reader to choose a suitable solution for their situation, including more secure options [5].

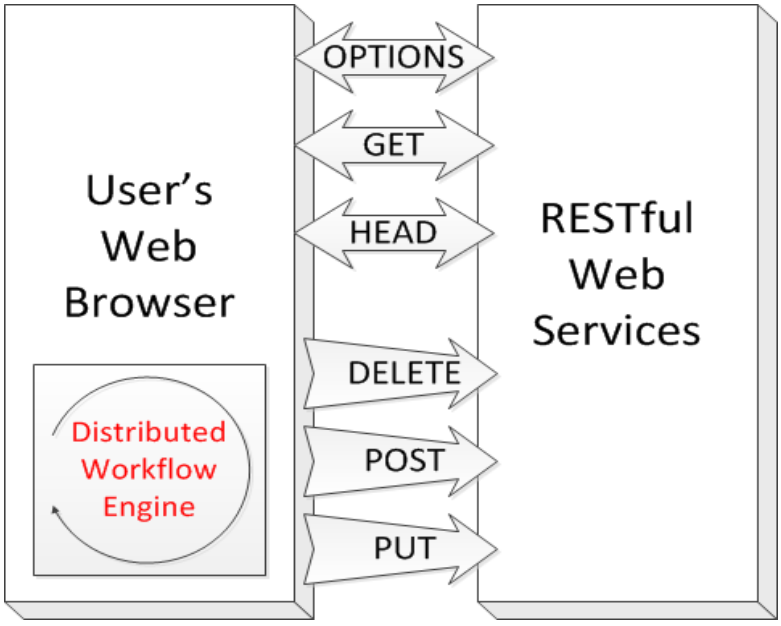

Figure 4: FlowCloud in context

2.4 Workflow Systems

Seeing as this work was referred to as a workflow system, a description of the same is in order. A workflow system is something that provides the ability to chain together operations and view or interact with the result of some flow execution. At its heart, this project provides a unique spin on more traditional workflow systems by changing the location where computation occurs, and by changing where and how data are shared. More specifically, it does so by putting more of the overall system execution in the client, due to the never before available power possible at that tier in the guise of the web. It also does so by incorporating the web browser itself, which is much different than traditional workflow systems. Many exist which run on massively clustered systems, at large scales which are either massively horizontal, vertical or both. These typically include a web based view, but none include the browser as the execution component directly [6]. A brief introduction to the existing dominant workflow systems will now follow to give a complete view.

2.5 Kepler Workflow System

Kepler is one of the better known and more widely used workflow systems, and has a long history as a server side workflow system in the biomedical domain [7]. Like most of the other workflow systems, Kepler provides a core execution engine, and it additionally provides a rather comprehensive graphical user interface which is used to design workflows themselves. Kepler relies on a heavy weight core

and fat desktop client to achieve its results. It can be used to create large server side workflow solutions on a larger scale than most of the other systems, and has been used to excellent effect by high profile projects such as OpenTopography [8]. In the case of OpenTopography, it powers the entire back end processing system.

2.6 Taverna Workflow System

Taverna is another large scale workflow system that provides both an engine and a graphical workflow creation user interface. Where it differentiates itself however is in the web service space. Taverna was one of the first to focus heavily on web centric variations, and included a web service API as well as a web based flow creator to utilize its execution environment. Taverna has been very heavily used throughout Europe on a wide array of projects, but has a strong following in the bioinformatics space as well [9].

2.7 Other Workflow Systems

There have been an array of web-based systems, which conceptually accommodate the notion of a workflow; however all of these as of today rely on a sizable component external to the core client to perform their actual execution, rather than executing or optimizing for execution by the web browser itself. A few of these will be discussed at length later in related work, as they are more closely related than the general overview covered above. The wide adoption and various

approaches illustrate how effective this model for execution abstraction can be, and show that it provides value to a myriad of processing problems. Regardless of the model or solution, the same concept of abstraction provides value, that of focusing user effort on the workflow as simply and visually as possible, while eliminating the need to know or interact with the lower level internals. Users spend time doing their analyses, rather than setting up processing systems and associated infrastructure, which is immensely effective and efficient compared to alternatives. On that note, it is imperative to briefly discuss processing systems more generally, which is what will be covered next, as that is a common alternative.

2.8 Processing Systems Overview

In the distributed systems processing arena there a number of other types of processing systems that have gained wide spread use in the past few years, and have been popularized by very widely used services such as Google. This includes things such as MapReduce [10], the algorithm behind Google's search. Apache Hadoop, which is an extremely heavily adopted framework that implements MapReduce. It has been adopted by the BigData movement as a front runner and chosen reference implementation [11]. In this sense, the term processing systems refers the whole spectrum of systems at various scales designed to read data, do something to it, and either display a result, return modified data or similar. This particular domain is largely what has driven adoption of cloud computing as an infrastructure, as these processing systems thrive as large distributed systems,

which leads nicely into a brief discussion of that topic for the sake of background details, which will follow next.

2.9 Cloud Infrastructure Concepts

The first and most important thing to keep in mind while discussing cloud concepts is to outline the various types there are, and identify which of those is relevant to the proposed solution. At a high level, there are three conceptual kinds of cloud infrastructures. In decreasing order of complexity, the three top-level types are (1) Infrastructure as a service (IaaS), (2) Platform as a service (PaaS), and (3) Software as a Service (SaaS). Infrastructure as a service replicates what is typically done either in internal computer labs or in off-site data centers, wherein virtual systems mimic the typical physical systems, but the effort and kinds of work required to create and maintain these sorts of virtual systems is identical to that of physical system setup, with the exclusion of racking and stacking hardware, as seen in figure 5. The act of building and maintaining a system though is more or less identical [12]. As a distributed system the cloud plays the single biggest role in allowing this work to exist, so an effort of due diligence will be applied to explaining the necessary pieces so that the foundation and its role can be thoroughly understood relative to this project. The designer of such a system utilizes virtual resources that look more or less identical to actual physical ones. Without a layer of abstraction, the amount of effort is on par with that of a traditional physical approach of the same sort. Virtualizing the physical resources in this manner

certainly has benefits however, chief among them, the benefit of easier overall system maintenance, which is a win for systems but not necessarily software.

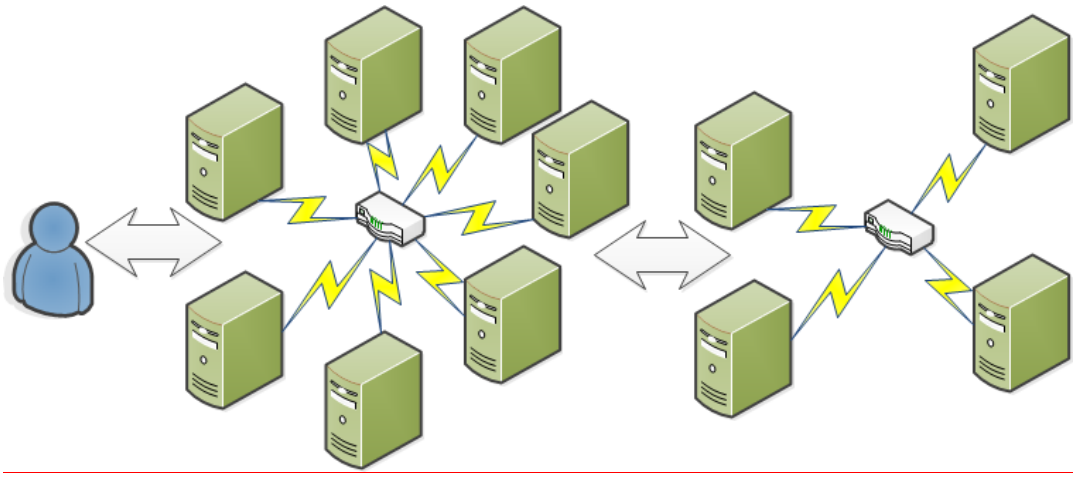

Figure 5: Infrastructure as a Service, virtual interaction same as physical

A platform as a service infrastructure on the other hand, starts one abstraction layer higher, and hides many of the system level details [13]. This sort of infrastructure typically provides an API level integration point for an application developer, while hiding all of the system level details from the same. This abstraction comes at the cost of flexibility, but can allow for faster creation of a system as a whole. Figure 6 show the way the user must interact with the pre-defined platform, on top of which they build their system or application, within the limitations of said system, tying them to that platform and limiting knowledge transfer. There are a host of these sorts of platform offerings available today, including Google's AppEngine [54], Heroku [55], CloudFoundry [56], Engine Yard [57], and OpenShift [58] just to name a few of the more popular options today.

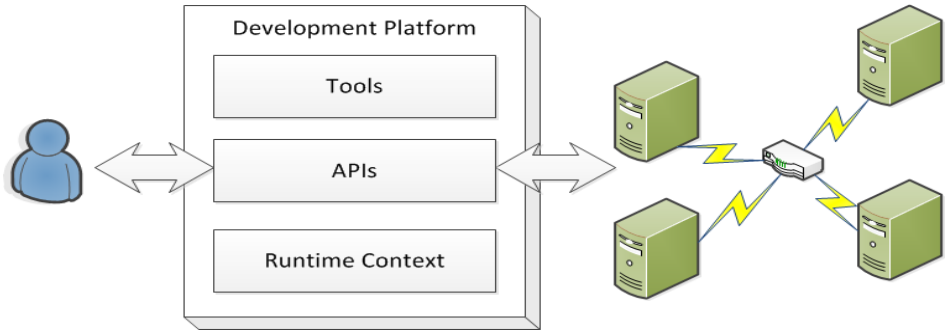

Figure 6: Platform as a Service, slightly higher level of abstraction

Software as a service infrastructures bring things up to an even higher level of abstraction, and hide both the underlying systems and any integration points of the middleware, and simply present an application, as shown below in figure 7. Many people use Software as Service applications in the cloud every single day, when using software such as Gmail or Google docs. As a user or developer, you use solely the application and know nothing of the systems or software supporting the application. These require the least effort on the part of the developer, but result in the most coupling to the platform and application directly, and make any applications built on them entirely reliant on their existence, as well as leaving the eco system more difficult to change out. For this reason, many services, Google's included, provide some sort of data migration capability. This work has no such issues despite providing the software as a service option. The workflow definitions can be uploaded, downloaded, and executed in different situations as desired. The focus is solely on building the workflow, executing it, and exposing a meaningful result, which is transferable independent of where the workflow executes, or the

underlying infrastructure doing the computation. Despite the focus on client side execution, this consideration is still very important when considering longevity.

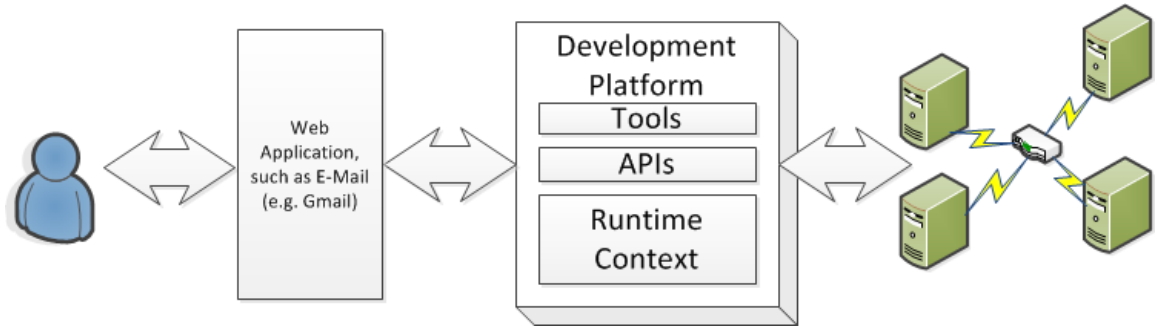

Figure 7: Software as a Service, where it sits in the spectrum

It is imperative that one have a clear understanding of the various types and levels of cloud environments when discussing where things fit together, and it was for that reason the previous descriptions were given. This work would be a viable component in any of these types of architectures, but in different ways and for different reasons. In an IaaS scenario, it would be more of another component in a larger workflow system where more traditional workflow systems run at the infrastructure level and are used as the underpinnings of a large scale distributed work flow system [14]. In such a context this work would be not be different than, or more useful than any other workflow system already in existence. Regardless, figure 8 illustrates why this model is not ideal for this project. It would result in several instances of the web based application running as the infrastructure without a clear way to interact with the system, or for the instances to coordinate with one another. This model will simply not fit this application's intended uses.

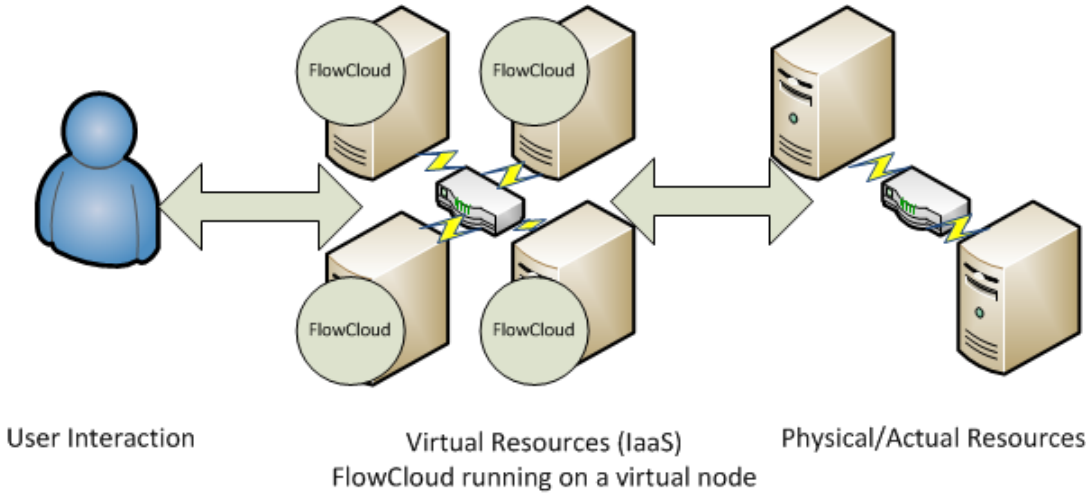

Figure 8: This work in the IaaS context.

In the PaaS context, this work could be either the workflow system itself, or a front end and in road to a larger scale workflow system of any type behind the scenes. Whichever of those models one would choose in the situation, the solution is still not ideal because it requires more effort and expertise than necessary, and again is no different than existing options, which figure 9 illustrates below.

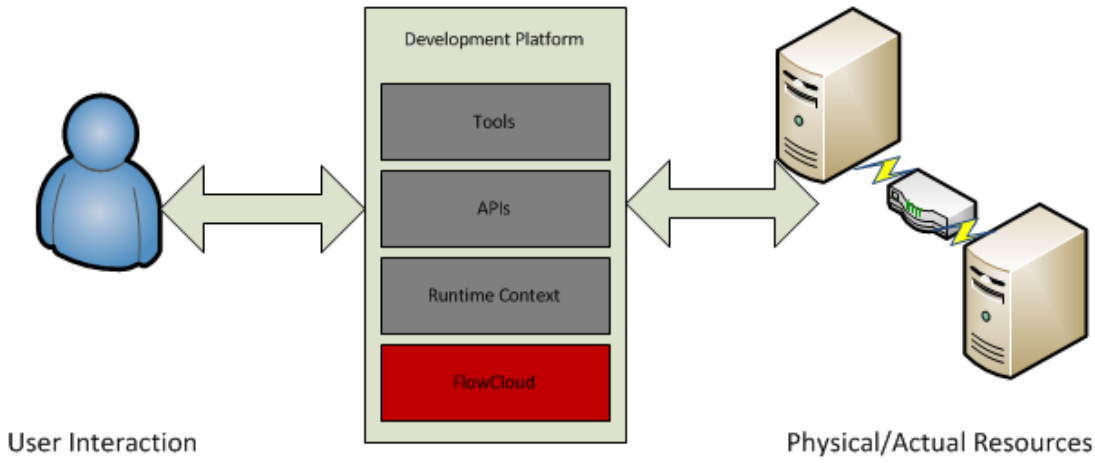

Figure 9: This project in the PaaS context.

In the software as a service (SaaS) scenario, this project would be more of workflow system in its entirety, without a more traditional distributed system or computing cluster in the mix behind the scenes potentially. The end user interacts with the workflow creation capabilities to define their workflow, execute it, and interact with a result. As shown in figure 10, this is the ideal scenario because the user neither knows nor cares about anything but the application, with which they are interacting to create and execute their workflow. This intelligent client approach is different than other available options because of how much occurs in the client directly, compared to the existing workflow system solutions.

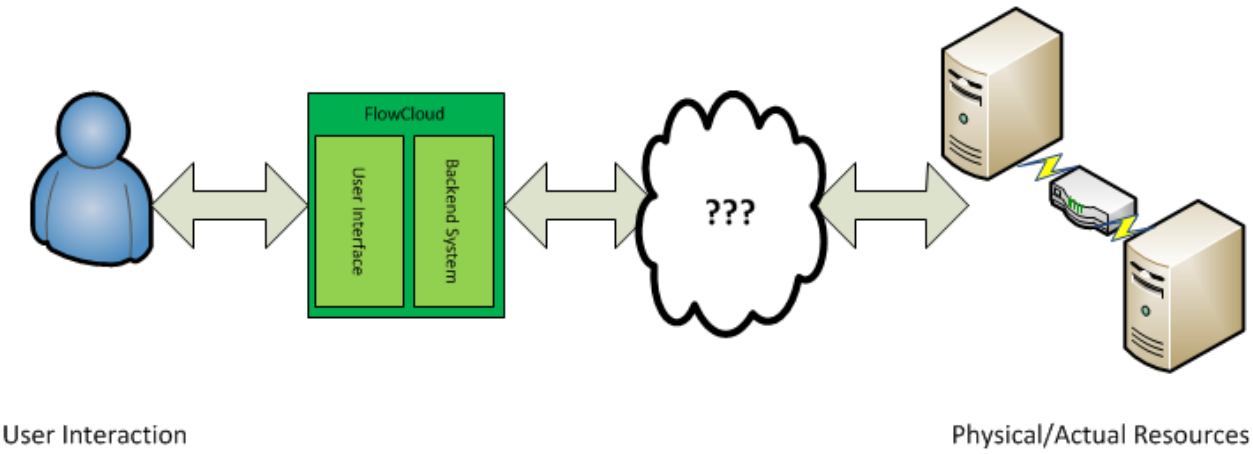

Figure 10: This project in the SaaS context, the ideal solution.

2.10 Web Browser Scripting Languages

Since this project is about maximizing the web browser based client, it is critical to clarify the available options for programming such a system. As a web based application, that means there is a limited selection to choose from. Another

area for potential innovation has to do with the incorporation of user-provided scripting, within the confines of existing web browser scripting languages. Specifically, what is meant here is allowing a user to quickly and easily supply their own algorithms or plugins, to accomplish complex tasks not originally envisioned by this work. For example, this work's limited scope only allowed for the creation of a handful of operations. However, if a user wanted to incorporate some additional similar capabilities that could just be executed against their data, then a plugin capability provides that. For example, statistical operations, more complex math functions, and even custom science data algorithms. For this to be viable and not require a dramatic amount of work interacting with other browser plugins like Flash, this means JavaScript. Specifically, the ability to execute user defined and uploaded JavaScript snippets against data obtained as part of workflow execution. The rise of JavaScript is indisputable [15], and as such rather than avoiding it or attempting to supplant it with something that achieves the same things conceptually, again for example by something like Flash, it should be embraced as a first class language. One such project embracing and taking JavaScript to a whole new level is Node.js. There are large scale enterprise ready frameworks written in and for the JavaScript language which serve domains previously only covered by much heavier weight and more traditional programming languages and environments [16]. There are back end frameworks such as the aforementioned Node.js that provide the means to encapsulate business logic in both a browser executable context as well as a server side one, using the exact same runtime environment [17]. All that said, for this work, the user will have the option of

supplying JavaScript snippets that will operate on data as part of the workflow. Workflow creation, execution and visualization merely require the user of the web application to operate inside of a web browser. This requires both intercommunication and workflow definition, which takes a modern approach as well, and will be discussed next.

2.11 Data Interchange Formats (JSON)

For the proposed solution a standard and well-recognized interchange format is utilized to ensure browser compliance and wide ranging support. Specifically, JavaScript Object Notation, or JSON, which is essentially a serialized JavaScript array that has a lot of similarities to XML in terms of how it's used as a data interchange these days. However, XML is a comparatively heavy weight solution, which is why JSON has rocketed to the forefront in terms of usage and adoption by modern projects, as seen in figure 11 below based on data from Google.

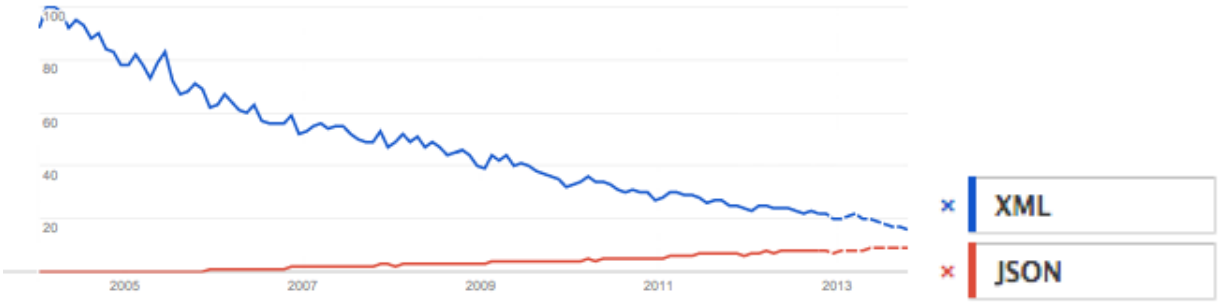

Figure 11: Trends of XML vs JSON 2004-2014 (projected)

JSON cannot be utilized for everything, but certainly many data sets, coordination and workflow definition. It is conceivable some raw data sets might be in other formats used in the science domain, so support for those will have to be considered and evaluated on an as needed basis. Some front-runners that will most definitely be needed are NetCDF and HDF. NetCDF already has available software that allows it to be dumped to JSON however [18], which would be preferable for consistencies sake. It comes down to this, there are three ways data needs to be shared; one, for the workflow itself, two, for the coordination between participants if there are parallel and distributed operations, which is really a subset of the workflow, and finally three, the science data being operated on. JSON can accomplish this on all counts [19] for the sake of this project. By way of illustration, here is one example data structure, to show what this looks like in practice, as used by this project currently, shown in figure 12 below. The name references a predefined operation which is a chunk of JavaScript code that operates on each data point, or the whole set, as intended by the operation. The example only illustrates the simplest variation of a workflow, but highlights the succinct description that results through using JSON. A number of web services suggest they support JSON, only to provide a simple JSON snippet that contains a URL to their native data format. It is important not to make the same mistake, as the web browser client cannot inherently interpret those other formats, or even do anything with them if it could. If JSON is the interchange, then it will be used completely, not just as a simple wrapper to describe other data sources. The example also clearly shows the flexibility in FlowCloud's workflow engine, in term the simple declarations it uses.

```
{
   "name": string,
   "details": string,
   "version": string,
   "operations":
   [
      "high_pass_filter",
      "low_pass_filter",
      "average",
      ...
   ],
   "data":
   [
      "input1":
      [
          {
             "url": string,
             "type": string
          },
      ]
      ...
   ]
}
```

Figure 12: Sample JSON workflow definition.

At its simplest, this is merely a list of operations to perform on a data set from a specified source, which in this case consists of some filtering and running an average on numeric data. Not all workflows need be distributed in nature, and the internals know based on the name of the operation where to look up modularized code to actually instrument the operation based on whether or not it needs to be (or could be) run in parallel if so desired, including on the GPU, as will be discussed later. The fact that the workflow encapsulates and hides these details is one reason it is simpler to use and why it is considered an intelligent client. These relatively simple examples described the mechanics of the engine for the sake of reaching common understanding, so a slightly more complicated example is in order and will follow shortly. In addition to the succinct way in which workflows are wholly defined, JSON can be heavily compressed as well, which is another benefit.

```
{
   "name": string,
   "details": string,
   "version": string,
   "operations":
   [
      "compute_nth_prime",
      ...
   ],
   "data":
   [
      "input1":
      [
          {
             "url": string,
             "type": string,
             "arguments": string
          },
      ]
      ...
      "output1":
      [
          {
             "url": string,
             "arguments": string
          },
      ]
      ...
   ]
}
```

Figure 13: Slightly more complex workflow sample

In the slightly more complicated scenario above in figure 13, the workflow is obtaining an Nth prime to compute based on index, doing the work, and returning the result to a centralized coordination location. Again, this isn't especially complicated, the point is to illustrate the capabilities possible, which in this case is a distributed computation of a list of N prime numbers, which may or may not be an improvement depending on the algorithm used. This does however show the potential in such a system, but it also highlights the need to choose problems to run on such a distributed system as carefully as possible for optimal efficiency [20]. This is no different than other processing systems and architectural decisions however; it always pays to consider the efficiency of applying a solution to a problem, and

determining if the solution makes any sense, case by case. So, all told JSON is the ideal solution because it is widely used currently, faster than other options such as XML [21], and just as human readable. As a data interchange in terms of both data itself, and workflow definition, the solution is a solid one within the goals and context of this project, as illustrated above. Beyond the interchange format used, the workflow system incorporates other advanced and leading edge capabilities, which will be discussed next.

2.12 Other Modern Web Browser Capabilities

There have been lots of web-based systems which conceptually accommodate very complex computational abilities. These are significant enough to discuss individually, which will be done later in the context of how they fit relative to this project, but solely to call out a couple specific examples, HTML 5 web workers [22], and the more recent WebCL [23]. Both will play an important role in the future of generalized in-browser computation in the opinion of the author. In the case of WebCL, the generalization of OpenCL and GPU computing will be possible in the browser as well, which is certain to have wide ranging potential. This project's goals potentially enable massive coordination across the web, of GPU computed results, which is nicely suited to problems where they can run in parallel to obtain data, process it, and then only need to share results or coordinate minimally. Now, with a reasonable understanding of all of the latest technologies incorporated into the project, the solution's approach will be discussed.

CHAPTER III

RELATED WORK

There are a variety of things that have some relation to this work in some way or another. Considering that the fundamental nature of this project is that it is a simplified distributed execution workflow system, albeit inside a web browser, there is some overlap in various ways with similar workflow style systems, but the fact that execution occurs in the browser client sets this work apart from these.

There are a few specific examples worth mentioning and discussing, as well as covering their similarities and differences to this work.

3.1 JavaScript MapReduce

As previously mentioned, MapReduce has been popularized in recent years by Hadoop, but has enjoyed a wide following ever since Google brought their variation of it to light. More recently there have been several [41] attempts at implementing a MapReduce framework for web based execution within JavaScript directly [42]. Granted, the solutions are extremely limited and simple [43], they do expose a rather solid proof of concept, and illustrates that it is possible to implement MapReduce within the browser client [44].

These endeavors are similar to this project solely in the sense that they provide a client side execution framework at a high level. Where they dramatically differ is that they only focus on the MapReduce distributed processing model, not more general computation, or even workflow creation itself, which is too limiting.

In short, the focus on the MapReduce model makes them applicable only to the set of problems that could be processed in such a way, and further, are in such a format to be processed from a web browser client.

3.2 Yahoo Pipes

Yahoo Pipes has been around for over five years now, since 2007, and embodies the same capabilities offered by this project, albeit presented in a much different way and targeted at a different use case, set of goals, and results [45]. Pipes allow you to create an aggregated mash up from multiple sources on the web, be the normal sites, photo sites etc. It is intended as a mash up system solely, for example, Pipes could be used to combine several RSS feeds and filter them, to geocode and display feeds on a map, or to be the underlying data source for graphical widgets pertaining to similar domains. In terms of similarities to this work, Pipes is a lot more focused, but does provide the same fundamental capability at a very high level. You create a workflow and execute it to get a result. That is where the similarity ends however, as the execution and coordination both occur on Yahoo's servers, not within your web browser, and the set of allowed operations is

extremely limited in terms of scientific computation (all but non-existent). Figure 14 below shows a quick example of what the interface looks like, provided for comparison to earlier discussions of this work

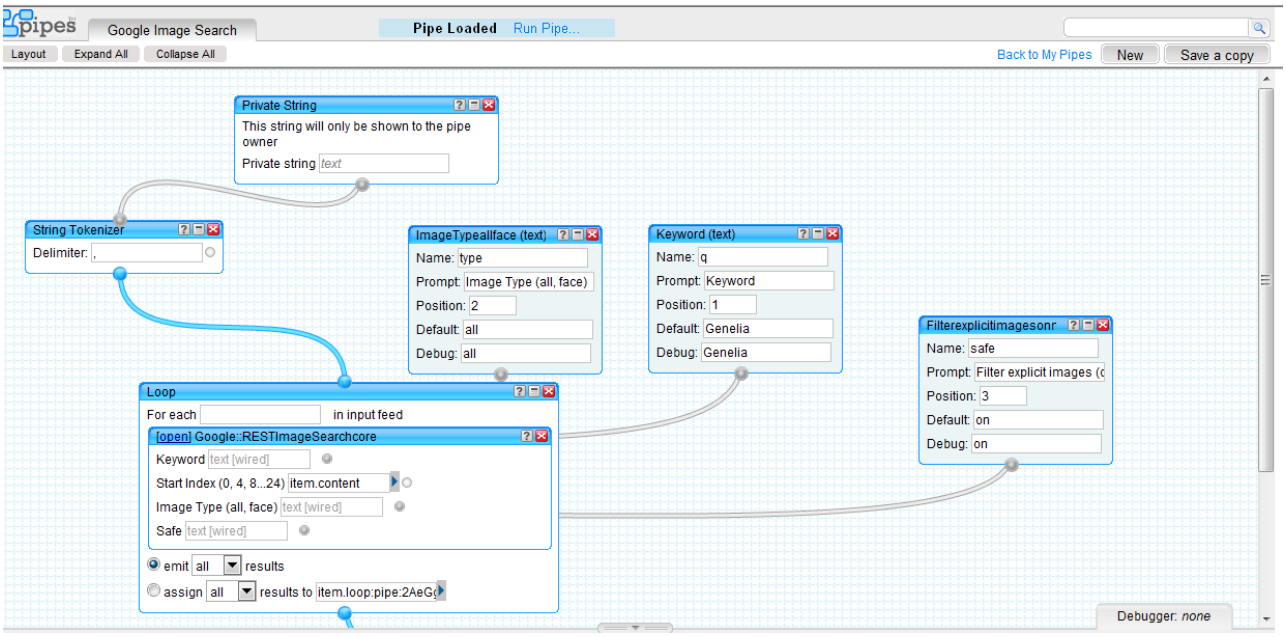

Figure 14: Yahoo Pipes example for Google image search

The proposed solution in this project is both more general and capable in terms of computation since it isn't limited to things Yahoo feels should be allowed.

3.3 Science Pipes

Science pipes bills itself as a very general tool for accessing, analyzing, and visualizing biodiversity data available online [46]. It incorporates scientific analyses, workflows, and even compares itself to Yahoo Pipes as well. As far as its

similarities to this project, it is fundamentally the same in some ways, in that it offers a web based application for creating and executing a workflow containing broad and general computations. Execution triggers a server side flow component though, not one which is running in the browser. The workflows themselves are equally easily shareable for collaboration, and equally powerful [47]. Figure 15 below shows an example of the interface for comparison. In this case, the example is fairly complex and related to bird species crossing boundaries. Being as it is a more science centric example than the Yahoo Pipes case, it is more directly comparable to this work and the FlowCloud implementation.

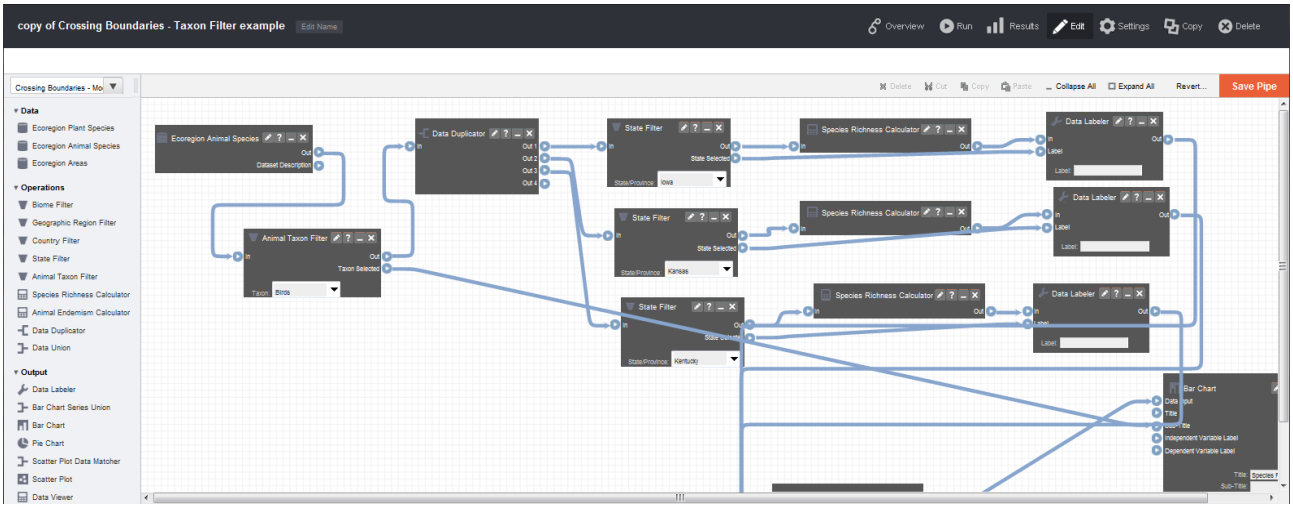

Figure 15: Science Pipes example

Again however, as with Yahoo Pipes, Science Pipes performs its computation on the server side only. In fact, its underlying execution engine is the aforementioned Kepler workflow system. No computation occurs on the web client directly. A web browser is only used to build the workflow itself and trigger execution. As with Yahoo Pipes, this difference is where this project stands out by

comparison. Since computation can occur in the web browser directly, the creator of the workflow is in direct control of the location of execution, you are not limited to the environment provided by the creators of the system. With that sizable difference though, Science Pipes is the most closely similar project to the work proposed here. Moving execution to the client, via the web browser, and incorporating advanced cutting edge technologies such as WebCL is where this work truly differentiates itself and offers up something unique. In addition, the ability to chain together server side systems such as Kepler, Taverna and others, also extends to Science Pipes, all of which could be used to create a much larger overall workflow with aspect of the execution taking place within the web browser, and others taking place elsewhere.

3.4 Node Workflow

Node Workflow [48] is a very interesting approach to the problem, but one that in my opinion is nearly identical conceptually to Kepler or Taverna, and only their server side components at that. It just happens to use different platform and data transport mechanisms, which in this case are Node.js and JSON respectively. Node Workflow allows you to define a workflow in JSON, as what it calls an abstract orchestration, and then operate the workflow through jobs within Node.js, which was described earlier. Kepler and Taverna's engines are based on Java, Node.js and thus Node Workflow is based on JavaScript. Kepler and Taverna rely heavily on XML for transport. Node Workflow uses JSON. In this way, Node Workflow is providing the same facility as both Kepler and Taverna, albeit in a

simpler to use, more modern web centric way. What it's lacking is a graphical component for building the workflows themselves, and a way to visualize a result, but to be certain it is most definitely a very valid approach to this class of problem, and could be used to accomplish some of the same things. Node.js cannot currently run within a web browser directly generally (exception below), so that relegates this solution to the server side, which is where the similarities to this project end.

3.5 Node.JS Inside Chrome

This relatively new software package [49] bridges two amazing technologies, but is limited to running inside of the Chrome browser only, so is not as practically viable as other solutions. It is similar in that it allows for complex computation in the client, but solely if that client is running Chrome, and has installed a special extension. From that point however, it would be possible to run very complex applications, many of which already exist for the NodeJS platform. The need to use Chrome and install the extension is also the limiting factor here.

As is evident in all the information provided above, there have been a number of approaches to this particular problem, and each has its strengths and weaknesses. Regardless, at this point you should have a clear picture of similar options available in the workflow ecosystem, and should have a decent understanding of how FlowCloud stands out.

CHAPTER IV

APPROACH

4.1 Problem Domain

This work touches on many large topics, so it is crucial that the scope and intent be clarified. This thesis work focuses on a very specific problem, while incorporating leading technologies as infrastructure and in so doing provides a unique solution to the problem of workflow processing systems, wherein it maximizes the capabilities in the client itself. It is critical one not confuse the infrastructural components with the solution itself. Thus, the problem domain is very specifically an intelligent web-client side workflow system based on web technologies, with a web based front end for creation, execution, visualization and configuration. At this time it has a limited number of processing capabilities specific to the science domain. Put another way, it is a cloud based workflow system, with a web based workflow builder, execution engine, and visualization suite, which does as much of its processing as possible in the users' web browser directly. That is the unique difference relative to other similar and related systems. This system is hosted in the cloud, exposes an end user interface via the cloud, coordinates via the cloud if and when necessary, but actual execution and computation takes place in the users' web browser directly. This is a dramatic departure from existing and traditional workflow processing systems, and generally speaking, is new territory

for JavaScript as well, which has had a bad rap and poor reputation until recent years. With that background in mind, it's important to define the confines of the approach, which will follow.

4.2 What This System Is

Considering the previous section describing and clarifying the problem domain, it is worth noting that this system provides three conceptual pieces. One, a workflow engine designed to run in modern web browsers; two, a web based user facing front end to execute workflows within that engine; and three, a web based user facing front end for designing workflows for execution. The second and third pieces could be accomplished a number of ways and have been undertaken mainly to easily utilize and highlight the intelligent engine that is core to the project, so the primary focus and biggest area for unique contribution here is the engine itself. The other two pieces are necessary to form a functional whole, so must exist, but are not new or unique in any way. In the past, these two aspects of a workflow system have been done as standalone desktop applications, web applications, and even mobile applications more recently [24]. Execution occurs inside the web client, coordination occurs in the cloud, if desired for parallel operations, and could extend to other distributed processing models in the future. The work described in this paper provides general computing capabilities, both on the CPU and GPU, inside the browser, staged by a workflow system. In short, this work creates a web browser client based workflow engine with a narrow specific domain in mind, and provides a

front end for defining and executing said workflows. It is a simple to use system for scientific pursuits, providing basic analysis capabilities for web accessible data, which empowers a user to utilize modern distributed and parallel processing capabilities without the need to understand complex system internals to do so.

4.3 What This System Is Not

The execution model outlined by this work is limited in scope by nature of the bandwidth limitations of the system overall. In other words, problem domains requiring low latency, high bandwidth sharing of data to conduct their computation need not apply [25]. As long as the link between them is a relatively slow network like the internet, this domain's problems will not map well into the space outlined by this project. As such, this system is not a replacement for more traditional larger scale workflow systems, or even more generally, clusters or similarly capable distributed systems that can provide both extremely high bandwidth, low latency connections between computation nodes. This project is envisioned solely as a cloud coordinated addendum to those, or a small scale simple workflow execution system on its own, with very limited processing scope relatively. To be clear, this system will not be viable for large data sets, nor high bandwidth applications or exceedingly complex algorithms. It is an intelligent client that is simple to use, and that exposes advanced capabilities in an easy to use way. It is extremely important to understand the intersection where this application has applicability. Again, this

work maximizes the intelligence and use of the web browser client, directly. The niche where this project's implementation excels is shown below in figure 16.

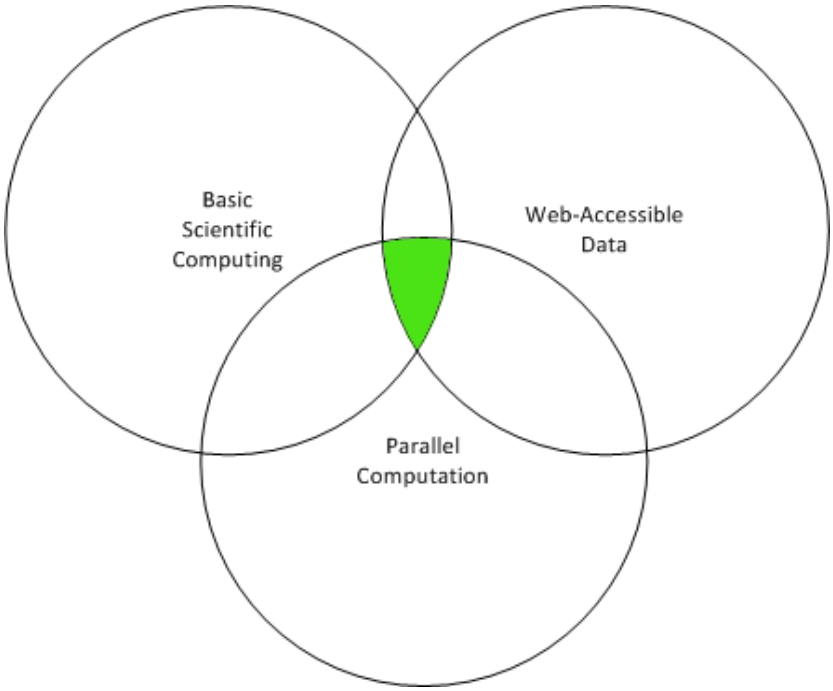

Figure 16: Overlap diagram showing ideal uses for this system

4.4 Proposed Solution Architecture

It is worth pointing out exactly what sort of architecture is being proposed and used for this work, in addition to the aforementioned discussion regarding what the system is, and just as importantly what it is not. As a component of a larger overall processing system, this work is still a cloud based solution, both in terms of where it runs, and what it exposes. It could run in the confines of either an IaaS or a PaaS infrastructure, and in turn expose a SaaS view of the underlying system regardless of the chosen deployment. Earlier figures simplified this illustration for

the sake of focusing on the architecture; figure 17 below will now explicitly identify all of the pieces end-to-end. In this case, the Platform as a Service used is Google App Engine. The intent of the implementation is to show all of the pieces discussed and provide an end-to-end solution utilizing everything discussed in this paper. However, the implementation is definitely not an enterprise ready application and likely contains bugs, and incomplete features. It does however provide all aspects of the solution discussed.

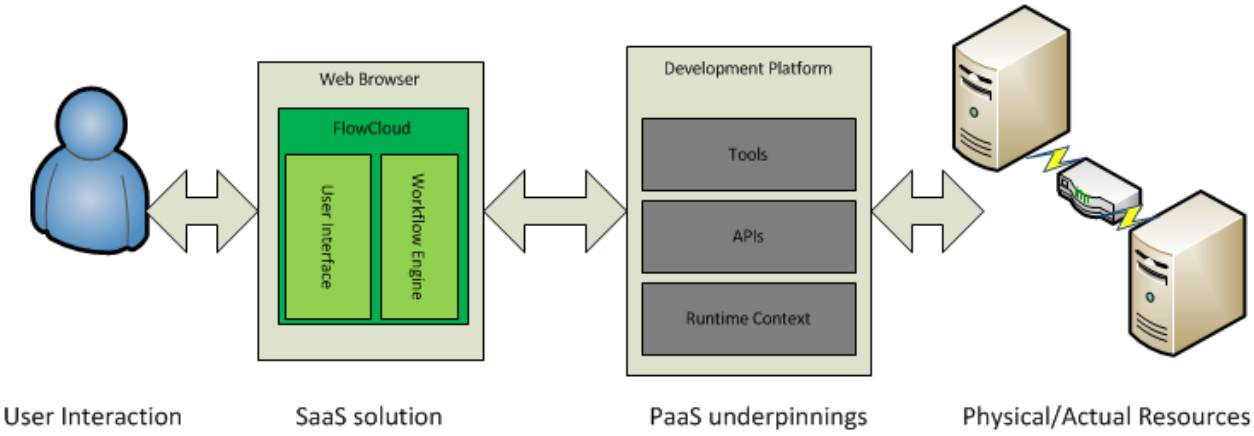

Figure 17: Diagram of the proposed infrastructural architecture.

4.5 Plugins and User Expansion

Another area of focus on this project is the incorporation of plugins that allow the end user to expand the system with their own features. This is limited in scope initially, as discussed earlier, and will only allow certain basic additional capabilities, such as user added math functions in the context of simple data processing. This project cannot possibly think of all the ways a user might want to manipulate their data, and it would be less than ideal if they have to export their

data and import it into something like MATLAB [26], Octave [27] or IDL [28], solely for the sake of simple processing functions which were not originally included by the project. Providing this sort of expansion serves this need, and is something that could improve to the point of allowing direct integration with one or all of the pieces of software just mentioned for even more complex operations.

CHAPTER V

METHODS

5.1 BUI vs GUI

Existing systems rely on thick client applications, which provide complex graphical user interfaces (GUIs) to allow one to build a workflow. A few solutions have a web application that provides similar capabilities, and in the context of a web browser client execution environment, coupled with cloud-based coordination, a browser based user interface (BUI) web application makes the most sense. The BUI is becoming more and more relevant, and may even see a massive increase in usage with things such as Google's Chrome OS growing in popularity [29]. The software industry is at a potential turning point, or inflection point of the traditional operating system versus the web browser [30]. Specifically, when considering a comparison such as Windows 8 to Chrome OS, convergence is occurring in this space. A similar comparison can be drawn between desktop and mobile platforms at this time as well. This presents a development challenge for software creators, and some opt for a web-based platform over native mobile implementations because of the ability it provides to target all the most widely used environments. With all of that in mind, the minimalist browser based workflow interface looks like the following diagram at present, as shown below in figure 18.

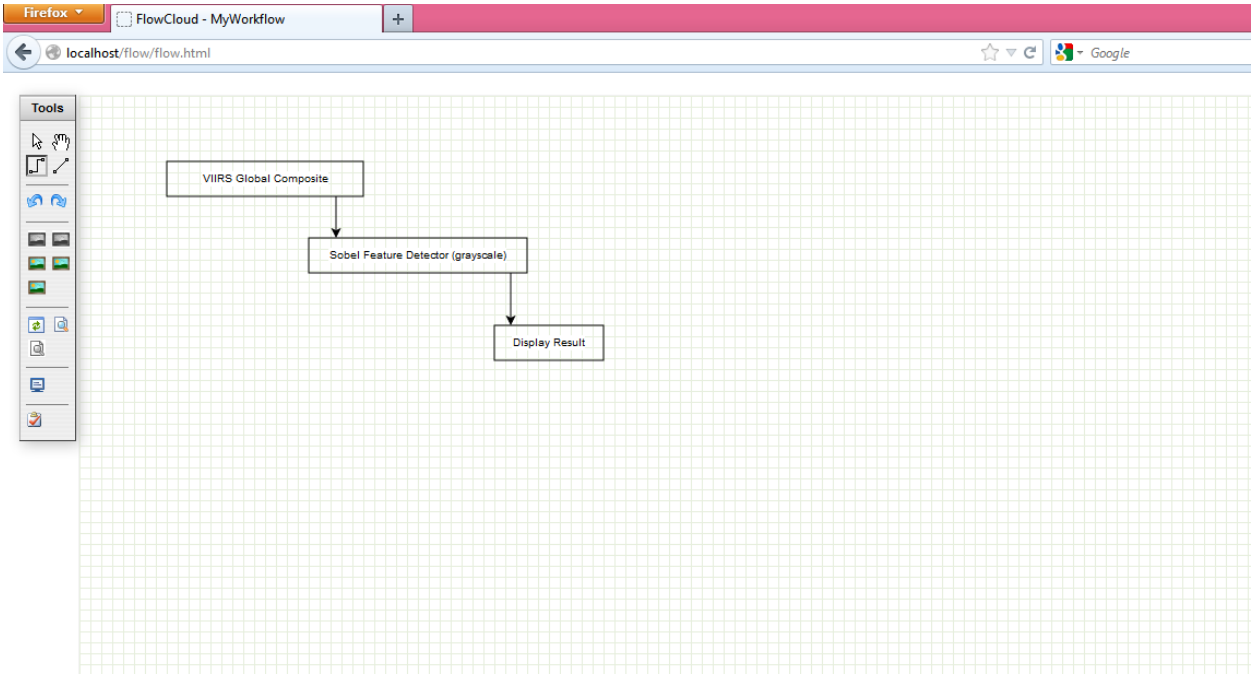

Figure 18: Minimalist Workflow Browser User Interface

5.2 Cloud Infrastructure (SaaS)

Cloud infrastructures were briefly discussed earlier, and it was mentioned that the SaaS model is being targeted for this project. As implemented, the targeted cloud infrastructure for this project maximizes the areas it intends to highlight and has identified as unique to the project. Initially, this is primarily statistical operations and the like, as well as some more advanced things like image feature detection, thus, the highest value integration and leverage point for this project in the context of a software as a service are the following three key areas; creation, execution, and visualization. This architecture allows the project to maximize computational throughput for the intended operations and use cases, while leveraging the tier of cloud infrastructure that simplifies usability for the intended

scientific user base. Data and the app live in the cloud, visiting the site downloads the code and client allowing for client side execution.

5.3 Web Browser Standards

The World Wide Web Consortium [31], or W3C, is the body responsible for defining various web standards, and in particular, HTML. As of now, the HTML version 5 standard provides several elements leveraged by this project, without which it would not be possible. HTML 5 is currently only a draft standard, and as a result has varying implementations across browsers. That is typically the case even with the finalized standards however [32]. Each web browser vender tends to have slightly different implementations relative to their interpretation of the standards. Some have historically and blatantly ignored aspects of these standards [33]. Without such standards though, and despite the inherent challenges relative to browser variations, this project has no hope of creating a consistent platform across both web browsers and operating systems in any other fashion. As such, these standards are a pivotal element underpinning this work.

5.4 HTML5 Web Workers

Web Workers essentially bring multi-threading to JavaScript applications [34]. In the context of this work, they allow for parallel CPU execution of workflow tasks, without having to be concerned with a thread model, or other complex

parallelization effort. A simple example illustrates their power below in figure 19, which is a complete example which implements a Fibonacci sequence tool [50].

```
var results = [];

function resultReceiver(event) {
  results.push(parseInt(event.data));
  if (results.length == 2) {
    postMessage(results[0] + results[1]);
  }
}

function errorReceiver(event) {
  throw event.data;
}

onmessage = function(event) {
  var n = parseInt(event.data);

  if (n == 0 || n == 1) {
    postMessage(n);
    return;
  }

  for (var i = 1; i <= 2; i++) {
    var worker = new Worker("fibonacci.js");
    worker.onmessage = resultReceiver;
    worker.onerror = errorReceiver;
    worker.postMessage(n - i);
  }
 };
```

Figure 19: HTML 5 web worker Fibonacci code example

Web workers operate on the principle of message passing, and the capability itself is thus event driven and asynchronous, for example as shown in figure 20 [35].

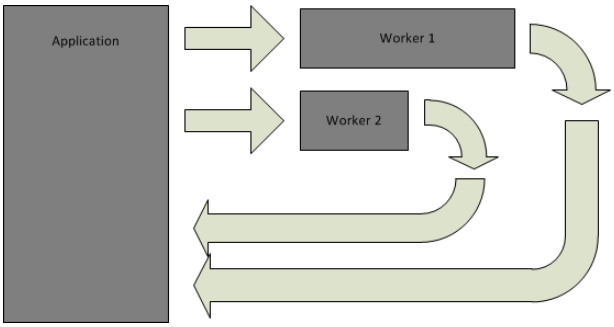

Figure 20: HTML 5 web worker asynchronous flow of control

Considering that the model chosen is that of a web application running on the client side, this leaves very few options for parallel computation, it is still possible however. Until recently, there was no good solution for this problem. Web workers serve this need well however, and prove ideal as the underlying model for a workflow system that is also inherently asynchronous potentially, and most certainly event driven. There are other options as well, that bring in the Graphical Programming Unit, or GPU, which will be discussed next.

5.5 WebGL

WebGL, for those unfamiliar with what OpenGL is, is a way of displaying graphics, which in this case is inside the web browser. WebGL will not be used directly, but it is an underlying necessity of WebCL, discussed next, which is in fact something which is utilized by the project. As such, a brief discussion of WebGL is still prudent, and follows. WebGL is the in-browser implementation of OpenGL, specifically OpenGL ES version 2.0, which supports shaders [36]. While not a fully capable variation of OpenGL, it is still extremely powerful and useful. However, unfortunately not all browser venders agree, and one in particular, Microsoft, has chosen not to implement the standard in its web browser. Microsoft have even stated publicly they have no intention of ever implementing it in their web browser, Internet Explorer [37]. That is the primary reason this area of the project is more experimental. By choosing a web application solution, by implication one of the goals is browser and operating system independence. Obviously, if a browser vender

doesn't implement a necessary feature, that browser isn't a viable option for that group of users. Therefore, the WebCL and GPU portions of the project should be considered more experimental, and will only function inside browsers other than Internet Explorer. Graceful degredation is possible with more effort, but at this time, it's all or nothing, so this work does not support Internet Explorer as an option. If the web browser does support WebGL and WebCL, the GPU options are usable. If the browser does not, the options are not even presented to the user. In addition to being necessary for some aspects of WebCL, WebGL could be used for graphically heavy plots and similar as well. Although presently the graphical components are not going to use WebGL, they most certainly could, for the sake of improving the interface in terms of usability and look and feel. Figure 21 below illustrates the WebGL rendering pipeline at a high level, to show where capabilities are available to the web browser. WebGL is farther along than WebCL, but still lacks the adoption by browser venders necessary to be considered truly viable as a platform independent option in its own right. The rendering pipeline is analogous to the desktop variation of the same that also allows shaders.

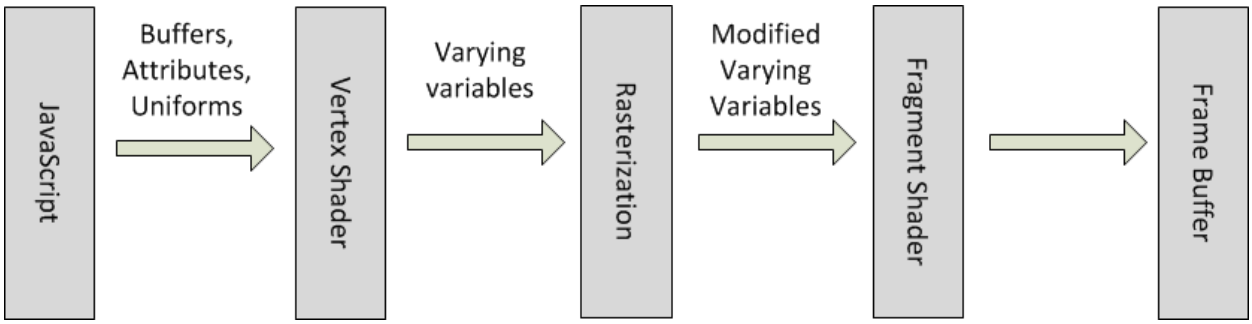

Figure 21: WebGL simplified rendering pipeline

5.6 WebCL

WebCL on the other hand is utilized by this project, and is a critical component of the intelligent client aspect of the same. WebCL brings heterogeneous parallel computing to modern HTML5 web browsers. It is essentially a JavaScript binding to OpenCL [38], accessible to web developers through the same extensive API, with 3 key limitations shown in table 2 below.

Web Browser Support	Firefox, WebKit (Safari, Chrome)
Programming Language Interfaces	JavaScript only
OpenCL Compliance	Only maintains "resemblance"

Table 2: WebCL's 3 key limitations

WebCL is an exciting new initiative announced in March 2011 [39], which adds many GPU programming capabilities to the browser. As of now, WebCL is only supported by a couple web browsers, but will be available more widely as time goes on, as has been the case with other previous standards in the past. As a unique contribution, the ability for workflow components to utilize GPU processing from a web browser is very compelling for several specific problems. A science user could create a workflow to operate on several large sets of data to compute a Fast Fourier transform for example, or do large scale image processing tasks such as feature detection. This aspect of the project interestingly combines horizontal and vertical scaling in one solution, but the focus remains on the intelligent client approach and

maximizing the vertical architecture therein. Recall however that this would only be viable as a solution for low bandwidth problems, as even with broadband penetration improving; by and large the web is still relatively slow in the United States specifically [40]. Regardless, the inclusion of WebCL into the mix is extremely intriguing on many levels. Figure 22 below shows a basic vector multiplication example from Nokia research [51]. Nokia's KernelToy is also utilized by this project [53]. The WebCL implementation began from the KernelToy project, which includes a Grayscale conversion Kernel. A WebCL Sobel Edge detector implementation was done as part of this project to provide a more complex example, and to illustrate the modular plugin capabilities described earlier. A kernel can be provided by a web service and loaded at runtime, which is illustrated by the grayscale and Sobel kernels implemented for the project. One is included in the workflow at creation, and loaded at workflow execution via web service.

```
<script id="clProgramVectorAdd" type="text/x-opencl">
  __kernel void ckVectorAdd(__global unsigned int* vectorIn1,
                            __global unsigned int* vectorIn2,
                            __global unsigned int* vectorOut,
                            unsigned int uiVectorWidth) {
     unsigned int x = get_global_id(0);
     if (x >= (uiVectorWidth))
     {
       return;
    }
    // add the vector elements
    vectorOut[x] = vectorIn1[x] + vectorIn2[x];
  }
</script>
```

```
function CL_vectorAdd () {

  // All output is written to element by id "output"
  var output = document.getElementById("output");
  output.innerHTML = "";

  try {

    // First check if the WebCL extension is installed at all
    if (window.WebCL == undefined) {
      alert("Unfortunately your system does not support WebCL. " +
            "Make sure that you have both the OpenCL driver " +
            "and the WebCL browser extension installed.");
      return false;
    }

    // Generate input vectors
    var vectorLength = 30;
    var UIvector1 = new Uint32Array(vectorLength);
    var UIvector2 = new Uint32Array(vectorLength);
    for ( var i=0; i<vectorLength;  i=i+1) {
      UIvector1[i] = Math.floor(Math.random() * 100); //Random number 0..99
      UIvector2[i] = Math.floor(Math.random() * 100); //Random number 0..99
    }

    output.innerHTML += "<br>Vector length = " + vectorLength;
```

Figure 22: WebCL code sample, vector multiplication

CHAPTER VI

RESULTS

6.1 Evaluation

Evaluating the results of this work will focus on tangible statistics, and will be based on comparisons to other existing systems, and will be conducted in a way that avoids uncontrollable variables such as network latency and the like. There are other subjective perspectives to consider as well, such as ease of use, and overall utility of the solution. That said, there are two vectors of result evaluation. One, a comparison of the workflow system itself relative to other similar systems, and two, performance comparisons within those systems to perform the same operations. First however, the general areas will be considered, since this work creates a generic workflow system, there is no clear cut way to measure it directly relative to other systems in a pure apples to apples fashion. The general nature of what is possible would be like comparing all possible programs to all other possible scientific computer programs, which is obviously not possible. What is possible is to compare similar systems, and the underlying direct performance differences of the technologies employed. This will illustrate and give a sense of how this system will perform more generally in actual use, relative to alternative means of accomplishing the same work with another solution.

First up is a comparison to the aforementioned Kepler and Taverna systems which are implemented in Java, since they embody most of the same goals, with different uses and platforms involved. For this comparison, a simple workflow was created in each system to obtain data sets of varying sizes and compute some basic statistical values, such as the average, the details of which are relatively clear. The normal JavaScript implementation is considered the baseline. From that comparison point, Kepler is ten times faster, while FlowCloud is twenty five times faster for this sort of data. Data acquisition is not considered in overall execution time.

As is evident from those results, performance is quite good for this sort of problem, and is well in line with the existing systems. The relative overhead of each system's workflow engine is negligible, and they are all well suited to this class of problem. It was discussed earlier that a unique contribution this work makes is the fact that the workflow engine, and execution, are occurring in the browser client. As such, this comparison is very important to show that is a viable model, and at least within the same ballpark as the more traditional workflow systems. Ultimately, it is well within an acceptable delta of the more traditional systems, and is indeed a viable execution platform.

Another area of unique contribution is the incorporation of GPU processing, also within the browser client. To prove the value of the

incorporation and use of WebCL, a comparison to the Kepler workflow system performing the same operation will suffice. For this comparison, a simple WebCL workflow was created which mirrors the traditional application as closely as possible. This is a simple application for the sake of comparing overall performance, and shows dramatic improvement over an implementation in JavaScript as well, as was apparent in Samsung's initial findings with WebCL where it provided 20-40x improvements [52], as shown in figure 23. The Samsung example below is provided as another way of illustrating the value of WebCL, as a way of establishing a pattern of high performance. The JavaScript variant manages 135 MegaFlops, compared to the WebCL variant's 5 GigaFlops. Clearly, WebCL is a valuable addition to the web developer's toolkit.

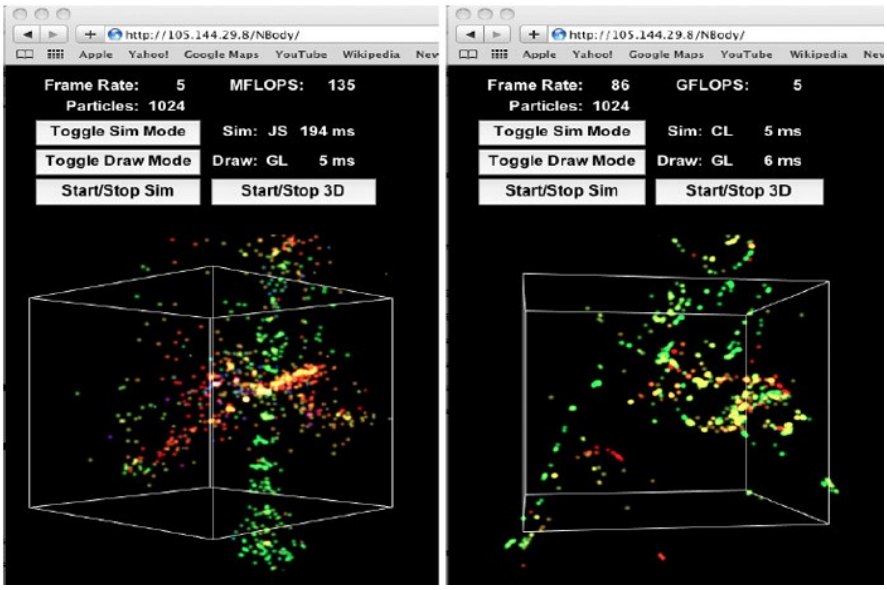

Figure 23: Samsung N-Body in JavaScript (left) vs WebCL (right)

These comparisons and evaluations are critical and important for the sake of illustrating the overall value of this solution and its most significant pieces. If it cannot perform in the same ballpark as other solutions, or provide useful tools, there is no valid reason for it to exist or be used. As will be discussed in the next section, and was shown at a high level in this section, the answer to that question, is it fast enough, is a resounding yes. This solution performs well, while providing new and unique ways of approaching certain problems.

6.2 Discussion of Results

At the outset, this work outlined several goals, to create a browser based workflow engine, with GPU processing capabilities, and client side flow creation tools. The combination of these things into an intelligent client as a whole system has resulted in a rather unique and interesting solution for certain domains. A discussion of the results will follow, and proceed from two perspectives. What follows is first, an outline of what is required to get the environment up and running and ready to use, to highlight the significance of the web application, and two, a performance comparison and discussion regarding a specific problem, satellite image feature detection.

First, an exploration of what is involved in getting both systems up and running and ready for an analysis by the user, the system outlined by this work, and the Kepler workflow system. Table 3 below outlines the steps required for

both in this scenario, assuming a user currently has an appropriate web browser installed on their computer; the table illustrates the zero to analysis state.

Kepler	***FlowCloud***
Download the application	Install the WebCL plugin
Install the application	Visit the FlowCloud web application
Open the application	Create and execute a workflow
Create and execute a workflow	

Table 3: Steps required for an analysis-ready system

There are two aspects to keep in mind here, the ease of installation and the ease of use. In the case of the former, it's clear the number of steps is approximately the same, so to find differences a deeper look is required. In the case of Kepler, the software, after download, expands to roughly 300 MegaBytes on the user's system which is much larger than any web browser. The WebCL plugin is extremely tiny, and as of the current version is only 218 KiloBytes. Combined, the FlowCloud solution and all of its prerequisites is substantially smaller than any other workflow system. Another area that differs dramatically, for the initial set up anyhow, is the time it takes for both solutions. Though the number of steps is roughly the same (one less for FlowCloud), they take a differing amount of time to complete. Installation time is another area where the Kepler solution takes much longer to complete because of its increased size. The first two steps of the Kepler process took hours to complete the first time due to

a slow download of the installer. FireFox on the other hand, over a common cable modem connection, took only seconds to install the WebCL plugin. The final step prior to readiness, that of opening the application, even takes drastically more time for Kepler. Specifically, opening the application and timing its startup to readiness and averaging the time it took, clocked in at 12 seconds. Starting FireFox and opening the FlowCloud application by comparison, similarly based on an average over five occurrences, came out to only 3 seconds. That warrants being repeated again. An existing system, Kepler, takes on average 12 seconds to start up to the point where it is ready to create or execute a workflow. By comparison, this project, the FlowCloud application takes on average 3 seconds to similarly be at a point where it's analysis ready, for workflow creation or execution. The net result is a clear benefit of the web application solution, and shows just how much lower the barrier to initial use is for a user, allowing them to quickly focus on the analysis at hand. In this regard, this work is at a clear advantage to the existing systems, even those that focus on client side applications in the case of workflow creation and execution, and illustrates the value of the chosen cloud model.

Usability is another matter entirely, but is not something this work can realistically evaluate, as it is a very subjective topic. As such, the fact that both systems, Kepler and FlowCloud, provide a list of operations and the ability to drag and drop them to create a workflow, they will be considered more or less equivalent in this sense. Whether that is entirely accurate or not will vary from

person to person. In this project, an effort has been made however to simplify operations and bring them up one additional level of abstraction. As a result, even though the concept is the same, the level of knowledge and effort required by the user will arguably put this work ahead in a usability comparison, at the expense of flexibility. Kepler provides a set of tools that operate at lower level and are thus more generic. This project provides them at a slightly higher level, namely, the act of obtaining an image from a web service for Kepler entails dragging a web service actor to the workflow, defining the URL, and incorporating it into the overall workflow for later execution. This project by comparison exposes the widget as the image itself. It requires more effort and programming behind the scenes, but results in a simpler easier to understand interface arguably. The end user does not need to know about web services or URLs, or where or how to get their imagery with FlowCloud.

With those details in mind, it should be clear that the solution provided by this project, for the intended use cases, is an improvement from the usability standpoint. The primary goal of this project is to provide a solution that maximizes the intelligence and capabilities of the client. Minimizing the level of effort required by the user is another avenue that serves that goal in a slightly different way. It allows someone to use the system to its fullest potential without knowing a lot of unnecessary and complex details.

The next area for result discussion involves looking at a specific use case and comparing this project to the Kepler system doing the same task. As stated before, this project was implemented to include all of the things discussed in this paper in an end-to-end fashion, to prove out all of the pieces and claims, so a problem was chosen to highlight that, while still allowing for a direct comparison to an existing system performing the exact same task. Namely, feature detection (detection only, not extraction) of satellite imagery, in this case a global composite from the VIIRS instrument on the Suomi NPP satellite. This task was explored with this project as well as Kepler, starting with figure 24 below which highlights the relatively simple workflow required to achieve the task in Kepler. This was a coincidence, as Kepler happened to already include an excellent integration of the ImageJ graphical library, and the means to call and script its behavior with a macro. The only downside to this is the additional knowledge required to achieve a result.

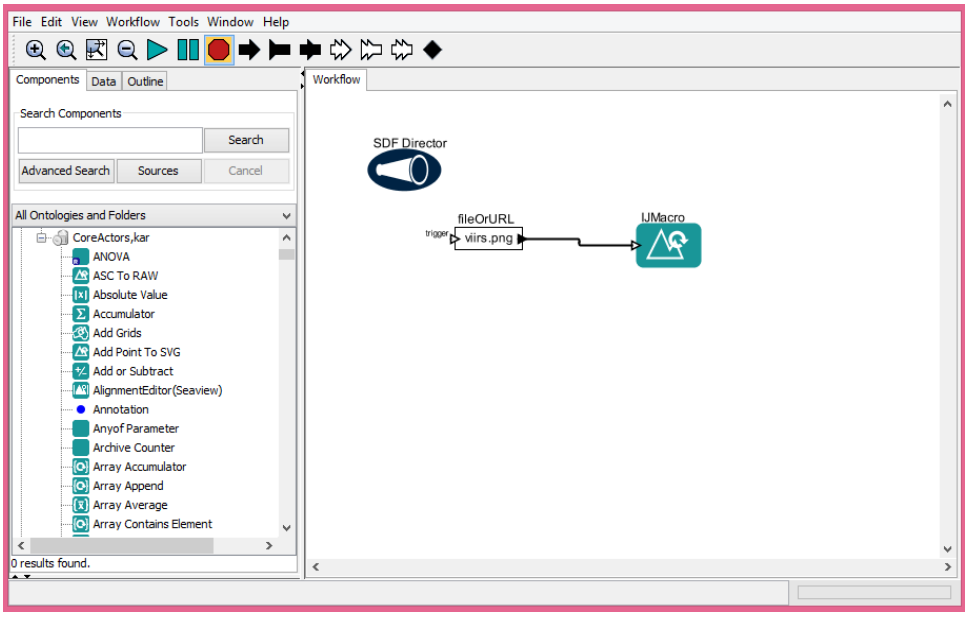

Figure 24: Sobel Edge detection workflow with Kepler

As you can see, this particular task within Kepler is relatively simple, because Kepler includes hooks to call the ImageJ library. Attempting to do something similar with Taverna was not possible unfortunately, as it is not typically used for these sorts of data. It could certainly have things added to do so however, it just was not possible immediately out of the box. When executing the workflow in Kepler, the ImageJ tool panel opens up for use, and can be scripted with the IJMacro Kepler actor, as seen in the workflow in the figure above. This tool panel is shown below in figure 25, which illustrates the range of capabilities provided by this library. As the statistics show later, the workflow may certainly be compact and simple, but it comes at the expense of performance. Loading the image alone takes a substantial amount of time, as does the edge detection itself, as seen below.

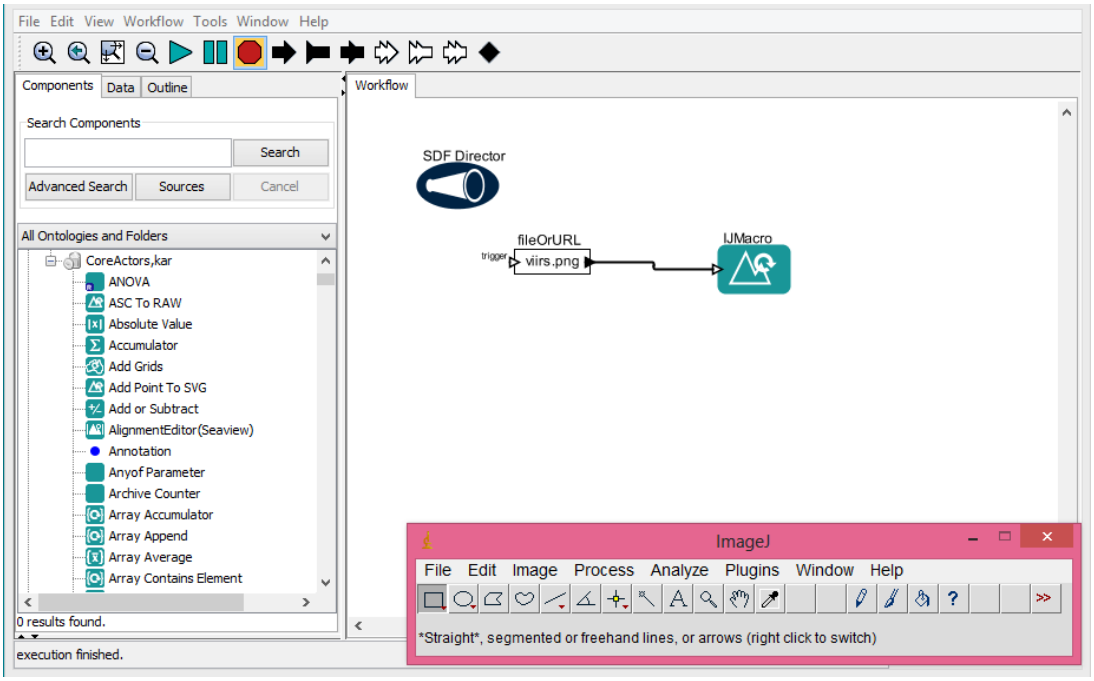

Figure 25: Kepler ImageJ panel

The real effort in this particular case is the scripting of ImageJ with the Kepler actor. This also strongly highlights the difference between this project and something like Kepler, which as you can see with this example, requires a fair bit of knowledge and effort to do a relatively simple analysis. Even though the workflow is small and simple, it requires a lot of knowledge to actually use. Figure 26 below shows what the scripting actor looks like that automates the ImageJ process during workflow execution, in which you can clearly see it requires even more knowledge, which in this case is the syntax required to interact with ImageJ in this macro fashion. For a simple two actor workflow, that is a large amount of expertise and knowledge required. REST services, Kepler internals, and ImageJ macro syntax.

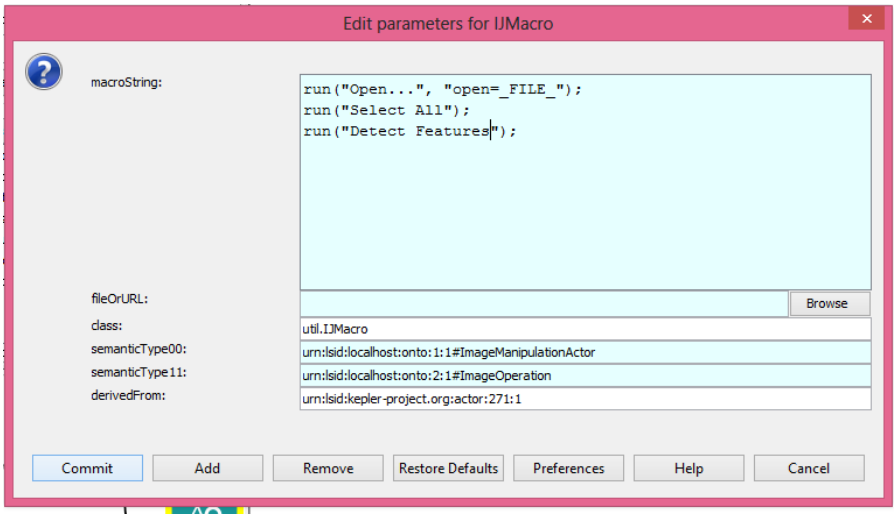

Figure 26: ImageJ scripting via Kepler actor

The scripting system is relatively straightforward, but it is yet another step required to get an end result, and one that is not entirely obvious or

transparent when something goes wrong. Following a successful execution, the resulting feature detection is shown in figure 27 below.

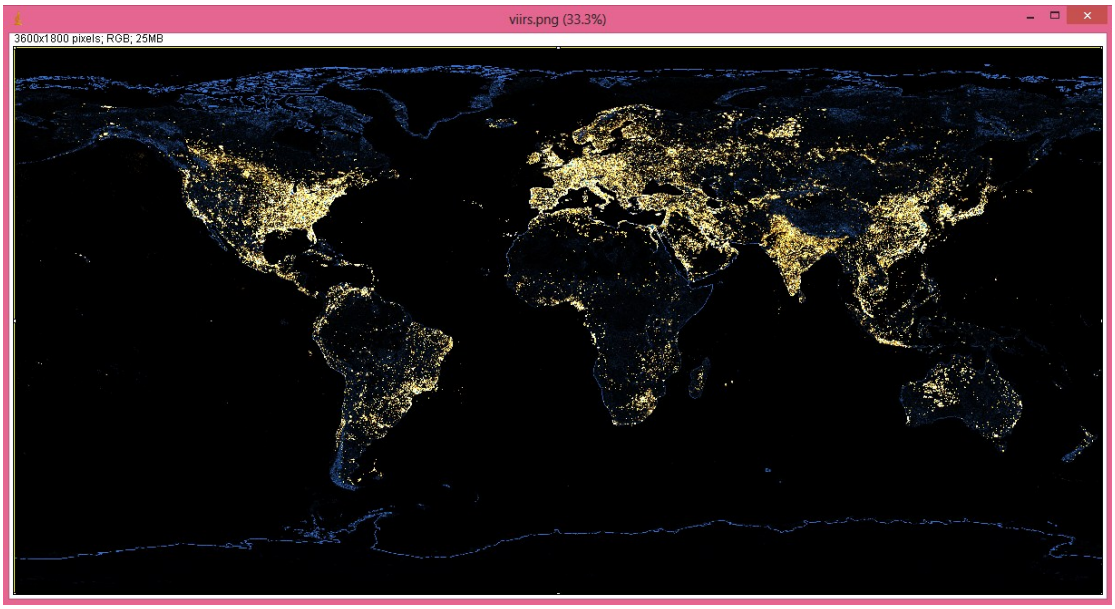

Figure 27: Sobel Edge detection result with Kepler and ImageJ

Following the walkthrough of this task through Kepler, it should be clear that the level of complexity and effort that was required is not inconsequential. For a scientist who wishes to quickly and easily perform such an analysis, this ramp up time is significant. It will become more apparent how significant later, following the comparison to this project's solution.

The same task with the implementation provided by this project has the same high level steps, but as will now become apparent, it is both simpler, and provides higher performance by incorporating the GPU into the execution process. Figure 28 below shows the same workflow as before, one which obtains

a global composite image via web service, runs a feature detection algorithm against it, and displays the result. It has one more step than did Kepler's.

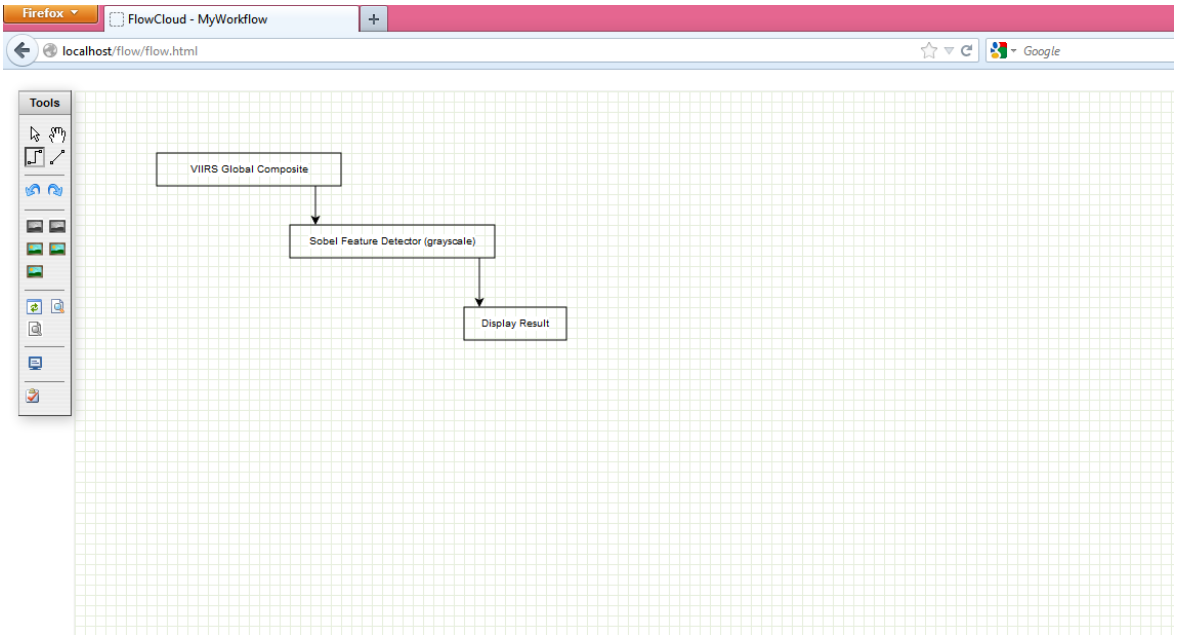

Figure 28: Sobel Edge detection workflow with this project

Continuing the comparison to Kepler, there is an important step not required by this project, and points out the value of the higher level of abstraction it provides. No scripting is required, no advanced knowledge of additional applications is required, the user simply created a three step workflow by dragging a source image in to the flow, followed by an algorithm to use for processing, and a way to view the result, and finally connected them in the desired execution sequence. No scripting, no ImageJ, no knowledge of web services, or the GPU. In this particular case this project's implementation obtains the image from a web service, knows based on the workflow that a GPU

based computation was chosen, and thus provides a result accordingly. The user only knew the image and the algorithm they wanted to perform, which highlights the value of the client incorporating as much intelligence as possible. Figure 29 below shows the result of executing this workflow within FlowCloud.

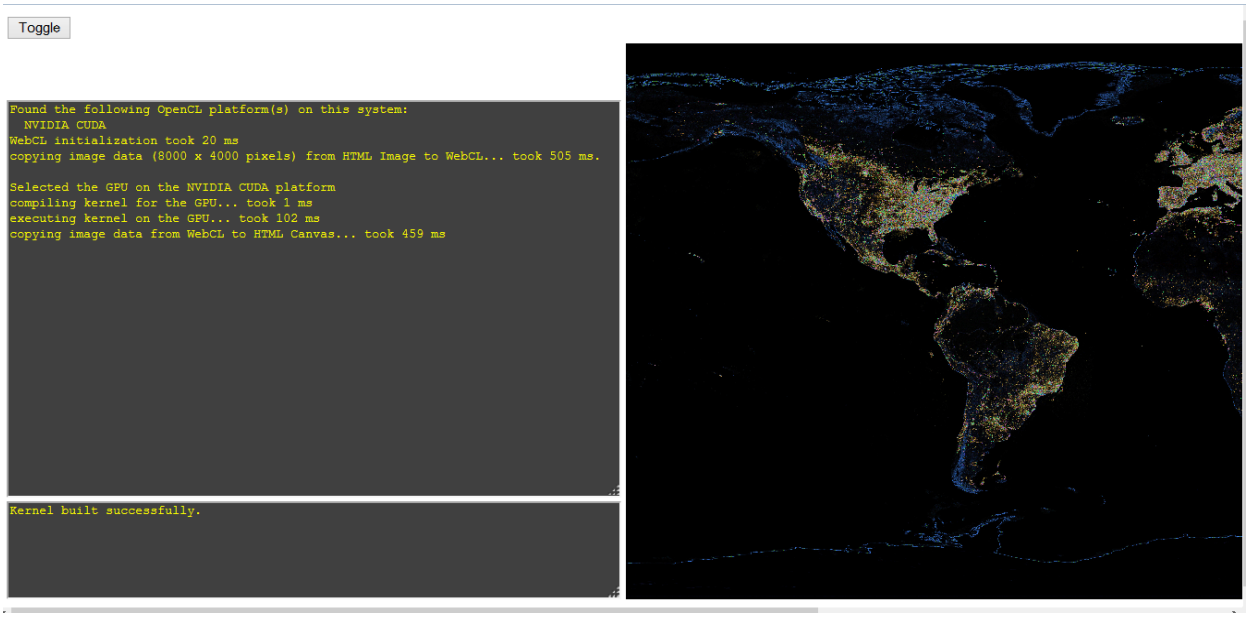

Figure 29: Sobel Edge detection output from this project

One challenge encountered in this area has to do with a race condition in executing the workflow and obtaining the image, due to the inherently asynchronous nature of web services and obtaining data. To illustrate, there is some amount of time in terms of a delay between executing the workflow and when the image is available by the browser to actually use. To work around this, a user prompt was added to ensure the data has loaded completely before attempting to run the chosen processing algorithm. This serves as a good reminder that this system is only applicable to imagery and data up to a certain

size, after which the time to load the imagery or data will render its usefulness more or less moot. In the case of the imagery operations, the user is supplied with a prompt, as seen in figure 30 below, but this could be improved upon and possibly even removed following some additional investigation, which outside the scope of this project.

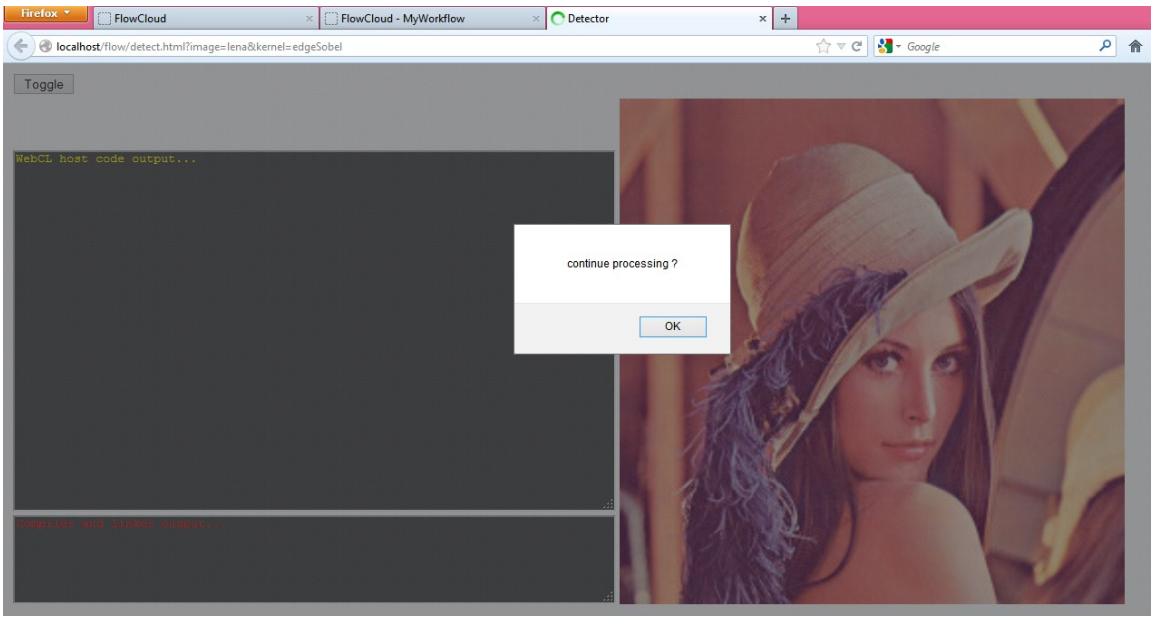

Figure 30: User prompted to continue processing

Once the user determines the image has loaded completely in the background, they can continue processing by clicking the 'OK' button to see their result, which for the example above is shown below in figure 31.

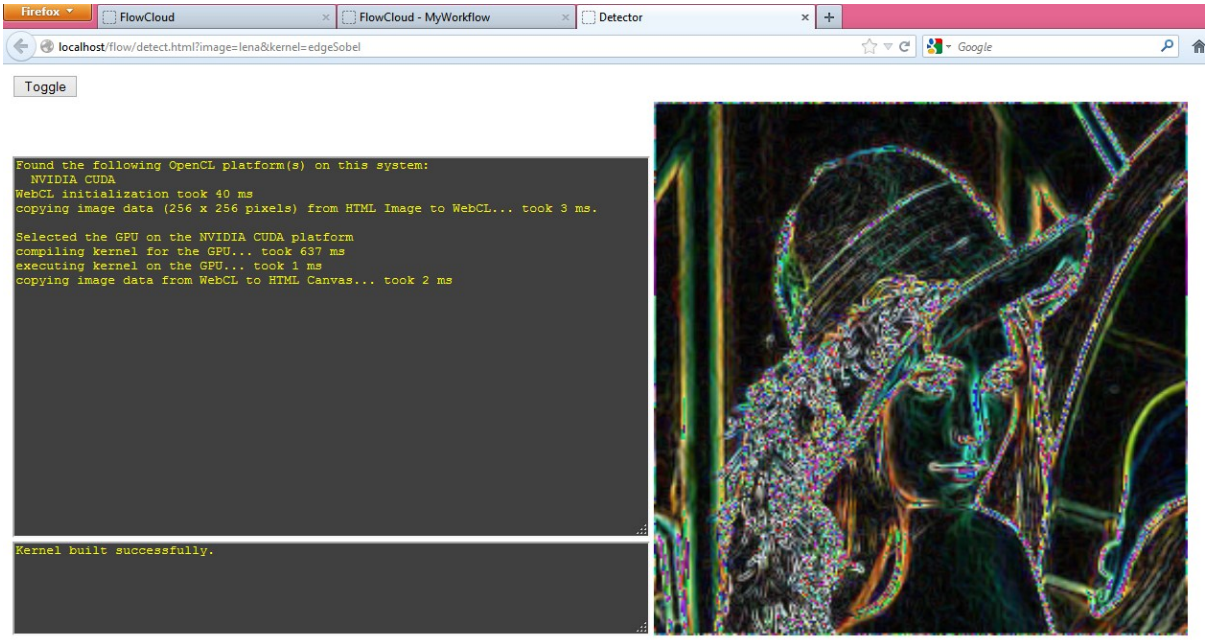

Figure 31: Processing completed after prompt continuance

It is this ability to abstract both data acquisition and complex operations such as GPU computation away to a simple model of move this image and these operation widgets around to achieve a result that contributes to this project's uniqueness. No knowledge of web services or GPU is required, pure focus on the analysis at hand is all that is necessary to obtain a result. This requires more effort in the creation of the workflow system itself, but dramatically improves the overall experience for a user. For most users, enough typical operations could be provided, but the plugin ability discussed before allows advanced users to go a step above and beyond when desired to expand the functionality of the system in new ways. It should be extremely clear at this point that the system provided by this project has achieved its goals. It is extremely easy to use, and provides a very capable and intelligent client that allows for creation, execution and

visualization in the guise of a workflow system, all without requiring the user to know a single thing about the technical details, including web services and GPU processing for example. Yet, it incorporates those and many more technologies to provide a maximally intelligent client within the confines of current web standards. A deeper exploration will now be done to compare the performance of the two approaches, again by executing the same workflows that were shown above. Kepler and FlowCloud will again be directly compared below.

A comparison of the startup of each system was done above, so the next topic of discussion will focus on performance for each of the solutions, specifically related to achieving the result above. Considering only computation time, there is a dramatic improvement in the implementation provided by this project, as will be shown below. The focus for the performance comparison will consider the time the execution begins, until it completes. This will include time to obtain data, transfer it to necessary resources such as the GPU etc. For each system, anything that is required from the time the execution is triggered to the time it completes, as a direct apples-to-apples comparison. At least in the sense of the overall process and algorithm being executed, beyond that there are differences in the implementations. For example, the Kepler based solution is not a GPU based solution, by design, to highlight the benefits of using the GPU on the web even more. Regardless, table 4 below outlines the process for each solution, and they coincidentally have the same number of steps again, despite going about

achieving the result in entirely different ways. In a way, this is good, as it makes the comparison less complex.

Kepler	***FlowCloud (this project)***
Obtain image from web service	Obtain image from web service
Open ImageJ	Copy image to GPU
Pass image to ImageJ for processing	Load/execute algorithm kernel on GPU
Instruct ImageJ to detect features	Copy resulting image from GPU to local
Get and view result	View image result

Table 4: Steps involved in each workflow execution

Using a Java profiler, in addition to capabilities provided by this project for its implementation, it is possible to inspect each step of the Kepler process and this project's, and profile performance each step along the way. Those detailed step by step comparisons will follow the higher level look at execution overall. First, below in figure 32, the specific performance comparison of the two systems is displayed. Several different images of varying sizes and resolutions were used to get a more accurate picture of performance across a wider range of data. This figure shows averages over five executions of the same operation. It is very clear in the comparison here that this project's implementation dominates in terms of performance. In fact, in most cases it is a full order of magnitude faster.

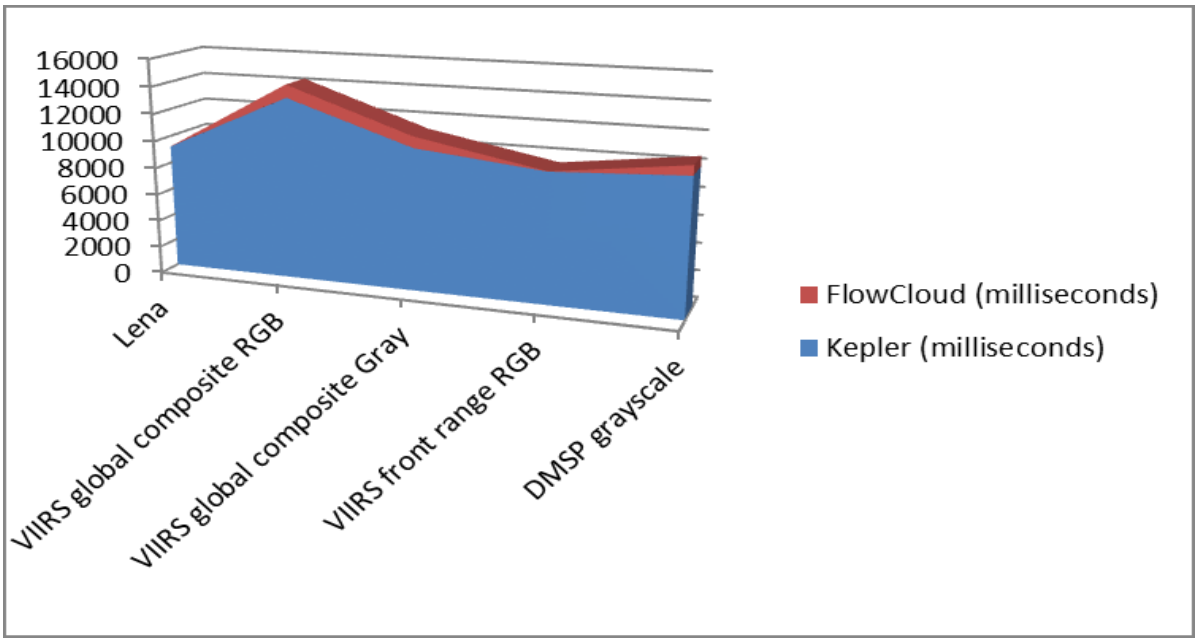

Figure 32: Overall performance comparison

In addition to that overall view, a specific example will be detailed farther. Namely, a step by step look at each piece of the computation for one image in particular, as shown below in figure 33 and tables 5 and 6.

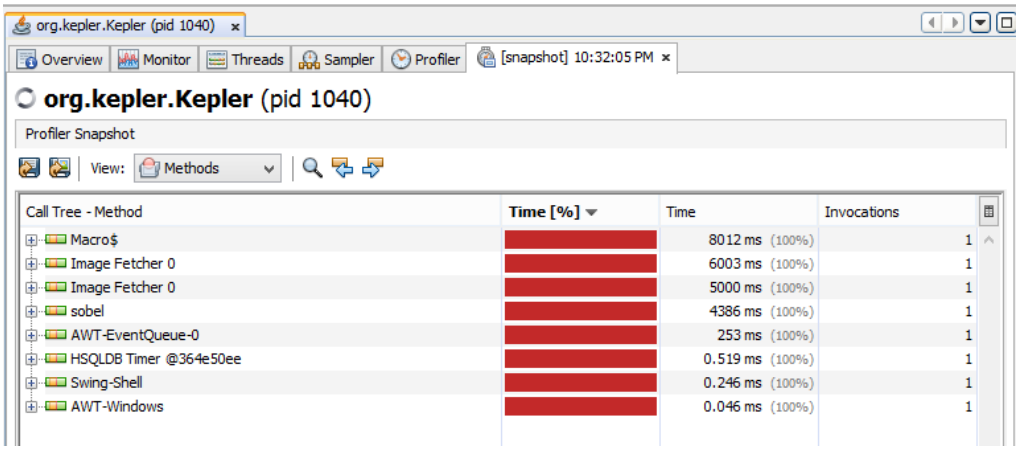

Figure 33: Profiling edge detection in Kepler

Kepler	***Time For Step In Milliseconds***
Obtain image from web service	5,000
Open ImageJ	8,012
Pass image to ImageJ for processing	253
Instruct ImageJ to detect features	4,386
Get and view result	0.046

Table 5: Step by step performance analysis, Kepler

By comparison, FlowCloud is dramatically faster at each step along the way throughout the entire workflow, for the exact same image and algorithm.

FlowCloud (this project)	***Time For Step In Milliseconds***
Obtain image from web service	68
Copy image to GPU	408
Load/execute algorithm kernel on GPU	79
Copy resulting image from GPU to local	308
View image result	1

Table 6: Step by step performance analysis, FlowCloud

Utilizing this system in such a way that the cloud is used a data hosting platform, and as much as possible of execution occurs in the client is clearly a viable model for a certain class of problems.

CHAPTER VII

CONCLUSION

In conclusion, this project has explored and shown that the web client environment is an ever increasingly capable general purpose-computing platform. Through new and existing frameworks and standards, said environment can operate in spaces previously reserved for super computers and large computer clusters. The intelligent client approach provided as a solution for the workflow system problem was clearly demonstrated as an extremely viable option. There are a number of potential future applications to consider, which will occur later in the future work section, but it is valuable to point out again that putting as much intelligence into the client, including things such as GPU processing, while using the cloud solely to store data and the application is an extremely effective way of providing tailored and customized workflow systems for certain domains. As implemented, this project brings the concept of a workflow system one level of abstraction higher than others available by not requiring the user to understand lower level programming details or interfaces. The implementation behind the scenes includes or uses advanced and high performance capabilities that allow the user to leverage them without even realizing they are in use, or that they even exist conceptually. The result is a system that exposes something very easy for a user to actually accomplish something, while doing said work in as fast and efficient a way as possible, as determined is optimal by the creator of the workflow system itself.

FUTURE WORK

The future expansion of this work would be most valuable in three areas. The first, coordination, could look at using a peer to peer model for coordinating work among larger systems. At present the centralized coordination via the cloud is ideal for certain classes of problem. Adding another means of communication that could maximize the relatively latent and limited bandwidth of the internet (relative to high bandwidth LAN options) could go a long way toward making this system applicable to solving more problems. The second would be expanding the available catalog of capabilities to a more complete set of scientific analysis centric operations. This work is not intended to replace packages like MATLAB or IDL, nor should it in the author's opinion, so not anything of that scope or complexity, just a more complete set of the mathematical and analytical tools used by various science domains. The third area of most value in future work is performance, specifically, caching both data and results for two purposes, speeding up computation, and allowing offline execution when and if possible for a given data set upon which a workflow is operating.

There are plenty of other areas where future work could be devoted, however the three mentioned above would build out this system as a potent and capable scientific tool, beyond the relatively basic analytical capabilities it currently provides as an end to end proof of the presented concepts.

BIBLIOGRAPHY

[1] D. Booth, H. Haas, F. McCabe et al., Web Services Architecture, W3C Working Group, 11 February 2004. http://www.w3.org/TR/ws-arch/

[2] C. Pautasso, O. Zimmermann, F. Leymann, Proceedings of the 17th international conference on World Wide Web, DOI: 10.1145/1367497.1367606

[3] L. Tang, Y. Zhao, J. Dong, High Assurance Services Computing, Specifying Enterprise Web-Oriented Architecture, DOI: 10.1007/978-0-387-87658-0_12

[4] P. Mazzetti, S. Nativi, J. Caron, International Journal of Digital Earth, RESTful implementation of geospatial services for Earth and Space Science applications. DOI: 10.1080/17538940902866153

[5] C. Jackson, H. Wang, Subspace: secure cross-domain communication for web mashups. DOI: 10.1145/1242572.1242655

[6] E. Deelman, D. Gannon, M. Shields, I. Taylor, Future Generation Computer Systems, Workflows and e-Science: An overview of workflow system features and capabilities. DOI: http://dx.doi.org/10.1016/j.future.2008.06.012

[7] C. Berkley, E. Jaeger, M. Jones, B. Ludascher, S. Mock, 16th International Conference on Scientific and Statistical Database Management, Kepler: an extensible system for design and execution of scientific workflows. DOI: 10.1109/SSDM.2004.1311241

[8] Krishnan, Sriram, et al. "OpenTopography: a services oriented architecture for community access to LIDAR topography." Earth Science (2011).

[9] Oinn, Tom, et al. "Taverna: a tool for the composition and enactment of bioinformatics workflows." Bioinformatics 20.17 (2004): 3045-3054.

[10] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[11] Nicolae, Bogdan, et al. "BlobSeer: Bringing high throughput under heavy concurrency to Hadoop Map-Reduce applications." Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on. IEEE, 2010.

[12] Lenk, Alexander, et al. "What's inside the Cloud? An architectural map of the Cloud landscape." Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing. IEEE Computer Society, 2009.

[13] Sosinsky, Barrie A. *Cloud computing bible*. Wiley, 2011.

[14] Yu, Jia, and Rajkumar Buyya. "A taxonomy of scientific workflow systems for grid computing." *Sigmod Record* 34.3 (2005): 44.

[15] Barros, Alistair P., and Marlon Dumas. "The rise of web service ecosystems." *IT professional* 8.5 (2006): 31-37.

[16] Paudyal, Umesh. *Scalable Web Application using Node. JS and CouchDB*. Diss. Uppsala University, 2011.

[17] Tilkov, Stefan, and Steve Vinoski. "Node. js: Using JavaScript to build high-performance network programs." *Internet Computing, IEEE* 14.6 (2010): 80-83.

[18] *"Unidata."* Software for Manipulating or Displaying NetCDF Data. *N.p., n.d. Web* http://www.unidata.ucar.edu/software/netcdf/software.html

[19] Nurseitov, Nurzhan, et al. "Comparison of JSON and XML data interchange formats: A case study." *ISCA 22nd International Conference on Computers and Their Applications in Industry and Engineering*. 2009.

[20] Othman, Ossama, and Douglas C. Schmidt. "Optimizing distributed system performance via adaptive middleware load balancing." *Proceedings of the Workshop on Optimization of Middleware and Distributed Systems*. 2001.

[21] Rodrigues, Carlos, José Afonso, and Paulo Tomé. "Mobile Application Webservice Performance Analysis: Restful Services with JSON and XML." *ENTERprise Information Systems* (2011): 162-169.

[22] *"HTML5 Rocks."* The Basics of Web Workers -. *N.p., n.d. Web* http://www.html5rocks.com/en/tutorials/workers/basics/

[23] "*WebCL*" Heterogeneous parallel computing in HTML5 web browsers -. *N.p., n.d. Web*. http://www.khronos.org/webcl/

[24] Kocurova, Anna, et al. "Context-Aware Content-Centric Collaborative Workflow Management for Mobile Devices." *COLLA 2012, The Second International Conference on Advanced Collaborative Networks, Systems and Applications*. 2012.

[25] Iqbal, Waheed, Matthew N. Dailey, and David Carrera. "Sla-driven dynamic resource management for multi-tier web applications in a cloud." *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. IEEE, 2010.

[26] Lindfield, George, and John Penny. *Numerical Methods: Using Matlab*. Academic Press, 2012.

[27] Quarteroni, Alfio, Fausto Saleri, and Paola Gervasio. *Scientific computing with MATLAB and Octave*. Vol. 2. Springer, 2010.

[28] Pallickara, Sangmi Lee, Shrideep Pallickara, and Marlon Pierce. "Scientific Data Management in the Cloud: A Survey of Technologies, Approaches and Challenges." *Handbook of Cloud Computing* (2010): 517-533.

[29] Taivalsaari, Antero, and Tommi Mikkonen. "The web as an application platform: The saga continues." *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*. IEEE, 2011.

[30] Sung, Eunmo, and Richard E. Mayer. "Students' beliefs about mobile devices versus personal Computers in South Korea and the United States." *Computers & Education* (2012).

[31] Vaughan-Nichols, Steven J. "Will HTML 5 restandardize the web?." *Computer* 43.4 (2010): 13-15.

[32] Ofuonye, Ejike, et al. "Prevalence and classification of web page defects." *Online Information Review* 34.1 (2010): 160-174.

[33] Mishra, Bharat, et al. "Study & Analysis of various Protocols in popular Web Browsers." (2012).

[34] Green, Ido. *Web Workers: Multithreaded Programs in JavaScript*. O'Reilly Media, 2012.

[35] Welc, Adam, et al. "Generic workers: towards unified distributed and parallel JavaScript programming model." *Programming Support Innovations for Emerging Distributed Applications*. ACM, 2010.

[36] Parisi, Tony. *WebGL: Up and Running*. O'Reilly Media, Incorporated, 2012.

[37] Ortiz, Sixto. "Is 3d finally ready for the web?" *Computer* 43.1 (2010): 14-16.

[38] Jeon, Won, Tasneem Brutch, and Simon Gibbs. "WebCL for Hardware-Accelerated Web Applications." (2012).

[39] Herhut, Stephan, et al. "Parallel programming for the web." *Proceedings of the 4th USENIX conference on Hot topics in parallelism*. 2012.

[40] Atif, Syed Muhammad, James Endres, and James Macdonald. "Broadband Infrastructure and Economic Growth." (2012).

[41] "Ben Nolan" Map Reduce in JavaScript
-. *N.p., n.d. Web* http://bennolan.com/2011/07/27/map-reduce.html

[42] "mapreduce-js" An educational MapReduce framework implemented in JavaScript -. *N.p., n.d. Web*. http://code.google.com/p/mapreduce-js/

[43] "Sevenforge, by Curtis Spencer" Meguro, a simple JavaScript Map/Reduce framework -. *N.p., n.d. Web*. http://www.sevenforge.com/meguro/

[44] "Igvita" Collaborative Map-Reduce in the Browser -. *N.p., n.d. Web*. http://www.igvita.com/2009/03/03/collaborative-map-reduce-in-the-browser/

[45] Casati, Fabio, et al. "Developing Mashup Tools for End-Users: On the Importance of the Application Domain." *INTERNATIONAL JOURNAL OF NEXT-GENERATION COMPUTING* 3.2 (2012).

[46] "Welcome to SciencePipes" access, analyze, and visualize the huge volume of primary biodiversity data currently available online -. *N.p., n.d. Web*. http://info.sciencepipes.org/about/

[47] "The NSTA Learning Center" SciencePipes: A World of Data At Your Fingertips -. *N.p., n.d. Web*. http://learningcenter.nsta.org/product_detail.aspx?id=10.2505/4/tst10_077_07_34

[48] "Node Workflow" A Workflow is Effectively an Orchestration. -. *N.p., n.d. Web*. http://kusor.github.com/node-workflow/

[49] "arunoda/chrome-node" NodeJS Runtime for Chrome -. *N.p., n.d. Web*. https://github.com/arunoda/chrome-node

[50] "Mozilla Developer Network" Using Web Workers -. *N.p., n.d. Web*. https://developer.mozilla.org/en-US/docs/DOM/Using_web_workers?redirectlocale=en-US&redirectslug=Using_web_workers

[51] "Learning WebCL" Lesson 3: Kernel Compilation -. *N.p., n.d. Web*. http://webcl.nokiaresearch.com/tutorials/tutorial3.html

[52] "WebCL" Samsung WebCL N-Body Simulation -. *N.p., n.d. Web*. http://www.khronos.org/webcl/

[53] "WebCL KernelToy" Nokia Research KernelToy -. *N.p., n.d. Web*. http://webcl.nokiaresearch.com/kerneltoy/

[54] "Google AppEngine" Google's Platform as a Service -. *N.p., n.d. Web*. http://developers.google.com/appengine/

[55] "Heroku" Salesforce.com's Platform as a Service
-. *N.p., n.d. Web.* http://heroku.com

[56] "CloudFoundry" VMWare's Platform as a Service
-. *N.p., n.d. Web.* http://cloudfoundry.com

[57] "EngineYard" Engine Yard's Platform as a Service
-. *N.p., n.d. Web.* http://www.engineyard.com

[58] "OpenShift" Redhat's Platform as a Service
-. *N.p., n.d. Web.* https://openshift.redhat.com/app/