Genetic Programming on the Cloud

Statement of Work

Una-May O'Reilly
Principal Research Scientist
Evolutionary Design and Optimization Group
Computer Science and Artificial Intelligence Lab (CSAIL)
Massachusetts Institute of Technology

1 Introduction

Cloud computing is rapidly being adopted as a cost-saving solution by enterprises because it offers demand elasticity and is priced on a usage basis. For small enterprises, second party clouds like Amazon's Elastic Compute "EC-2" or Simple, Storage Service "S3" are suitable options. Larger businesses may alternatively create a proprietary, private cloud for their organization.

Beyond cost effectiveness, cloud computing offers novel application and service opportunities. This is largely due to its so-called "infinite" capacity: a cloud delivers as much computing (storage and CPU cycles) as needed, just when it is needed. A cloud can power reactive, data streaming applications which are based upon Artificial Intelligence (AI) technology, such as machine learning, case-based reasoning, knowledge-based reasoning, planning, soft computing and evolutionary computation. It permits AI-based components to process information at a scale which empowers their top-level applications to deliver more accurate and more timely results for larger and more difficult problems. It enables agile and flexibly scaling AI-based applications for dynamic and real-time domains such as prognostics or condition-based monitoring.

It is rapidly becoming well understood how to get the most out of the cloud as an application technology platform for developing complex web services or database servers. This is because the cloud and internet have grown hand in hand. However, when one considers cloud-based Artificial Intelligence, a best-practices understanding of how to transition to the cloud has not yet emerged. This is because a lot of the technology supporting cloud application development is still itself being developed. Universities and industrial labs such as Microsoft, Google and Amazon are still conducting research into virtualization, cloud operating systems, cloud programming models, cloud programming languages and cloud application development environments and support. Many "nuts and bolts" details of massively parallel computing will be abstracted away from cloud application programmers. Theoretically this might make programming easier. Complex parallelism issues of locks and race conditions may be solved at a lower system level. However, because the cloud can migrate applications, a program must be, to some extent, oblivious to the architecture upon which it executes. This removes fine-grained performance control which programmers currently have. In general, the means by which cloud abstractions will be provided and the specifics of some of these abstractions are still open to invention and definition. This implies that the software engineering options related to how to develop, deploy and update cloud-based AI algorithms are expanding and in flux.

Also still resolving is the appropriate design of AI algorithms which execute on the cloud. Due to obvious circumstances, state of art, pre-cloud algorithms are designed and applied in a manner that ignores elasticity, massive computing resources and extreme scale problem complexity. They are often parallellized in a rote manner after they have been designed on a serial computational model. Unfortunately such porting frequently ignores the fact that inspired re-distribution of control and data or decomposition of program flow and logic can improve the capability of an algorithm. For example, island-based genetic algorithms compute faster and frequently more accurately.

Over the 4 years of this project we intend to closely keep in step with trends related to the two issues just raised: how to best software engineer on the cloud and how to design innovative cloud-based AI algorithms. Our attention will be guided by our specific goal of developing cloud-based genetic programming symbolic regression.

2 Project Conception

The Industrial AI Lab of the General Electric Global Research Center invents state of the art technology systems for data mining, information fusion, predictive modeling, optimization and decision support. It develops algorithms which extract useful knowledge from operational data and which create models for diagnostics, prognostics and health management. It generates technology roadmaps aligned with business strategy and visions plus identifies value stories for business processes and system transition. The lab's strength in adopting computational technology innovation trends has motivated it to adopt cloud computing. A cloud offers the Industrial AI Lab the potential to:

- build more timely and reactive systems in application areas such as engine and rail diagnostics,
- address problems of larger complexity such as large-scale balance of plant monitoring and power management,
- consolidate currently disparate software components into an integrated toolkit, development and service platform which can efficiently serve diverse applications by eliminating software redundancy arising from independent development efforts.
- share data across businesses to enhance broad application interaction, seamless service delivery and base capabilities
- centralize the experience and lessons learned concerning development and deployment of AI-based services.

To support its adoption of cloud computing, the Industrial AI Lab is interested in forging a relationship with a university research group involved in cloud AI research. This will advance the Lab's strategic need to encourage early-stage software development in cloud-based AI. Through such a relationship the Lab can broaden and strengthen its cloud knowledge base and stay abreast of AI-based emerging cloud technology trends in a cost-effective way. The Lab will have a means of informing and guiding research attention, at a critical time, toward issues it considers important. It can first hand observe how cloud-based AI algorithm design at the academic level is proceeding. It can increase recognition of GE's commitment to cooperative technical community efforts by supporting academia's policy of open source software.

The Evolutionary Design and Optimization (EVO-DesignOpt) group led by Dr Una-May O'Reilly at the Computer Science and Artificial Intelligence Laboratory (CSAIL), MIT has a research agenda at the intersection of evolutionary computation (EC) and cloud computing. In one direction, the group is investigating how evolutionary computation can extend the competence of the cloud technology stack. Dr. O'Reilly partners with members of CSAIL's computer architecture group which affords her, as an EC expert, a unique position to continuously stay abreast of advances in cloud technology. The advances also inform and influence her agenda to transform evolutionary algorithms on the cloud. One goal of her EC research agenda is to create a cloud-based genetic programming compiler. The compiler will auto-configure a customized genetic programming symbolic regression cloud-based system which can be integrated into a higher level AI application. The project vision includes supporting multiple heterogeneous genome representations with novel distributions of the problem's explanatory variables. These choices will be manually available to application developers or automatically selected for the developer by a compiler. They will be capable of interacting with an asynchronous, continuous computation system which flexibly scales to cloud-scale numbers of problem objectives and training cases. The computation will be tunable to variable solution accuracies and response time deadlines.

Because of these intersecting circumstances, the interests of EVO-DesignOpt and the Industrial AI Lab are obviously well matched. We thus propose a project to be conducted by Dr. O'Reilly at MIT under the sponsorship of the Industrial AI lab in partial support of her cloud-based genetic programming compiler agenda.

3 Project Proposal

Period of Performance and Personnel

The project consists of 4 phases each corresponding to a year of performance at CSAIL, MIT. The project will be led by Dr. Una-May O'Reilly. It will also involve a post-doctoral researcher in the first year, then a graduate student in years two through four. The project will allow the graduate student to work towards her/his doctoral dissertation. Based upon ongoing requirements, as the project matures in years two through four, additional effort of a post-doctoral researcher may be enlisted.

Project Outcomes

The project has 3 outcomes:

1. open source software
2. a continuous dialogue between Dr. O'Reilly's research group and the Industrial AI Lab
3. technical research

Open Source Software

One goal of the Industrial AI Lab is to foster an open source community centered on cloud-based evolutionary computation. An open source project will encourage some of the brightest minds in EC and multiply this project's impact. The multiplier may be in anticipated directions, outside the scope of this project, or, like many open source systems, it may reach beyond the vision of the original developers.

MIT will release open source software under an MIT software license at multiple time points throughout the project. More details about MIT License can also be found here. The time points will be jointly agreed upon by the Lab and MIT. Also by mutual agreement, project software will be transferred to the Industrial AI Lab. Transfer may assist with planning specific application oriented extensions.

The open source software will be promoted at a variety of forums including major conferences in the EC community. Dr. O'Reilly interfaces with ACM's Genetic and Evolutionary Computation conference (GECCO), and Dr. Piero Bonisonne of Industrial AI Lab interfaces with a variety of leading IEEE Computational Intelligence Society's conferences like WCCI, SSCI and CEC. Together both teams intend to invigorate and instigate passion for community open source development of cloud-based evolutionary computation.

Continuous Dialogue

A major outcome of this project is a continuous dialogue between EVO-DesignOpt, CSAIL and the Industrial AI Lab.

One aspect of the dialogue will involve the Lab guiding the technical developmental process toward areas of its interest and learning about the design challenges, potential resolutions and project progress from EVO-DesignOpt. We plan bi-weekly telecons to exchange this sort of progress information.

There will be approximate monthly visits from the Lab to CSAIL. These will be scheduled to align with seminars and events of related interest at CSAIL. The dialogue will keep the Industrial AI Lab appraised of trends in cloud technology both from an AI perspective and from the perspective of cloud system engineering.

Another goal of the dialogue is to identify future opportunities that are compelling to the Lab. EVO-DesignOpt will describe its other research projects that are related to the Lab's interests and introduce the Lab to potentially interesting new ideas around cloud computing.

Overview of Technical Research

This project will support Dr. O'Reilly in developing open source algorithmic building blocks and conducting basic cloud related research that further the goals of her cloud-based genetic programming compiler agenda.

We detail a plan for the technical research in Section 4 and use Figure 1 to present a visual overview. There are 3 activities during the first phase (Year 1) of the project, see Section 4.1: The major activity is to develop a basic genetic programming symbolic regression (GPSR) system for the cloud. A second activity is to develop a taxonomy of components that should be available when compiling customized GPSR systems for a wide array of AI applications. A third is to develop a detailed, complete functional specification of the cloud-based genetic programming compiler.

In the second phase (Year 2), see Section 4.2, we will compose use cases for the cloud-based genetic programming compiler. We will then select hand-in-hand one use case and a swath of components from the taxonomy (which suit the use case). We will develop the components to run on the cloud, synthesize them into a system, and hand tune the system to execute efficiently on the use case's cloud resource set. This will allow us to determine in what ways the hand customized cloud system is superior to the basic GP system we developed in the first phase.

In the third phase (Year 3), see Section 4.3, we will develop a methodology to automatically, rather than manually, synthesize a GPSR system from taxonomic components. We will start to build the compiler which, given its component library, application requirements and cloud resource parameters, automatically synthesizes a cloud-efficient GPSR system.

Finally, in the fourth phase (Year 4), we will enable the compiler's synthesizer to autotune and integrate experience learned from historical system performance (see Section 4.4).

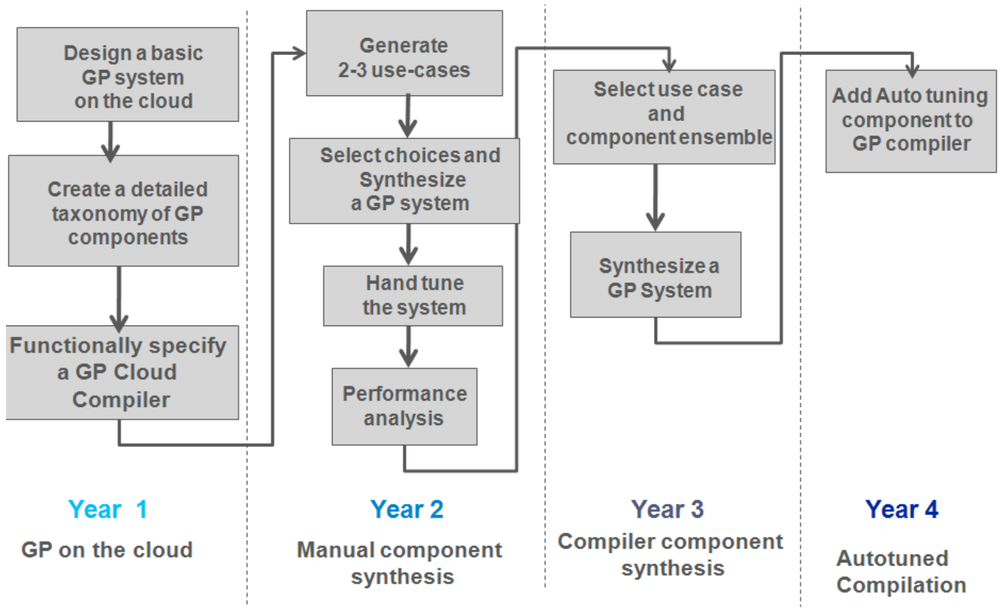

Design a basic GP system on the cloud
Create a detailed taxonomy of GP components
Functionally specify a GP Cloud Compiler
Generate 2-3 use-cases
Select choices and Synthesize a GP system
Hand tune the system
Performance analysis
Select use case and component ensemble
Synthesize a GP System
Add Auto tuning component to GP compiler
Year 1
GP on the cloud
Year 2
Manual component synthesis
Year 3
Compiler component synthesis
Year 4
Autotuned Compilation

Figure 1: Project activities, Year 1 through Year 4.

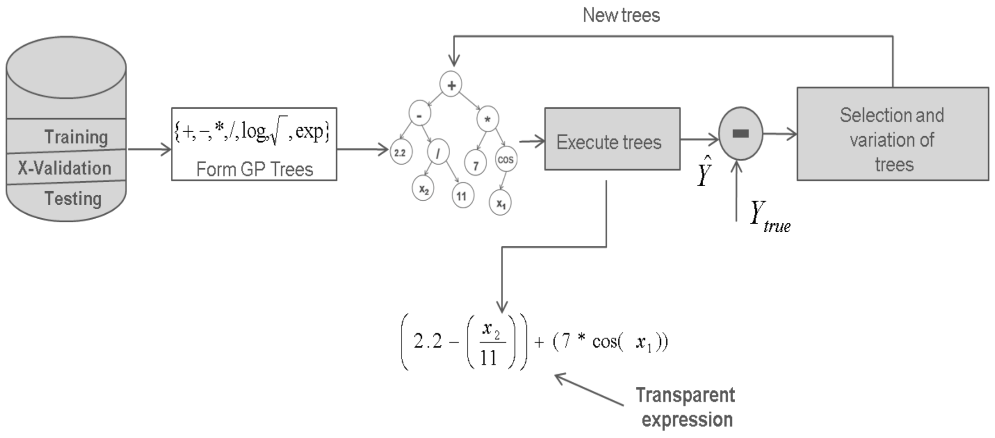

New trees
Training
X-Validation
Testing
{+, −, *, /, log, √, exp}
Form GP Trees
Execute trees
Selection and variation of trees
$\hat{Y}$
Y_{true}
$\left(2.2 - \left(\frac{x_2}{11}\right)\right) + (7 * \cos(x_1))$
Transparent expression

Figure 2: A genetic programming symbolic regression (GPSR) system. The algorithm outputs mathematical expressions that capture non-linear interactions among explanatory variables to generate a response. It can produce multiple solutions by exploiting its population based learning and is easily distributed and parallelized.

4 Technical Research Plan

Genetic Programming Symbolic Regression

This project focuses on developing quantitative models using a data driven machine learning technique for regression and modeling, called genetic programming symbolic regression (GPSR). GPSR is inspired by neo-Darwinian evolution and genetic algorithms. It generates mathematical models of diverse, flexible structure (linear, quadratic, non-linear, etc) using a conventional machine learning process of training, cross validation and testing. Its best evolved models are used to predict responses to potential observations. The mathematical structure of these models does not need to be posited *a priori* because GPSR is capable of learning both model structure and parameters simultaneously.

A conventional GPSR algorithm, see Figure 2, creates a set ("population") of candidate functional linear and non-linear models in the form of abstract syntax trees, using any quantity of elements from a set of arithmetic operators which typically includes addition, subtraction, multiplication, division, inverse; exponent, and natural logarithm. During each iteration of the algorithm, a parent population is selected by probabilistically choosing the better models. The population of the next iteration is then created from the parents via mutations and crossovers of their trees. In crossover, random sub-trees are exchanged between two parents preserving syntactic correctness while exploring semantic variation. In mutation, with a certain probability, a small number of nodes in one tree are changed.

GPSR has matured over a number of years with researchers developing a diverse set of features which have improved its robustness and effectiveness. For example:

- the refinement of accurate model parameters has been improved by a post-run linear optimization step.
- different representations, such as grammar-based integer vectors, machine instructions and assembly instructions have been invented. A niche for each has been outlined and demonstrated.
- parallelization on GPUs and clusters of the island-based distribution model has occurred.
- multiple measures to counteract bloat have been developed.
- multi-objective genetic programming (e.g. Pareto GP) has been developed.

Taking advantage of these advances, in this project we focus on developing a *flexibly factored and flexibily scaled* GPSR algorithm.

A *flexibly factored and scaled* GPSR algorithm:

- is a continually running solver providing a series of solutions, rather than a one time solution derivation
- efficiently trades off solution precision, solution accuracy and time to compute its solutions.
- consults application requirements and resource information (there will be self-aware chips and operating systems and application support from programming models) to self-adapt its work toward efficient use of resources while meeting functional requirements.
- is capable of different sets of work, can execute on different types of hardware with different genetic representations (and interpreters).

The GPSR algorithm can be flexibly factored and scaled with respect to:

- problem size (dimension)
- speed of computation
- scale of hardware, i.e., GPU, multicore, cluster, grid, cloud
- how much effort is used and time expended to compute a fitness evaluation
- training data volume
- objectives

4.1 Phase 1: GP on the Cloud

The major task in Phase 1 is to develop a GPSR system on the cloud. We will attempt to address the following questions:

- What parallel programming paradigms are best suited for developing a GPSR system? We will investigate a variety of parallel programming paradigms that can be used to put a GPSR system on the cloud. These include but are not limited to: Map-Reduce[1], rabbitmq [2], and BOINC. .
- What are the best options for distributing a population across multiple cloud compute nodes for the GPSR system?[1]
- What attributes specific to the parallelization and parallel programming paradigms are well suited to (a) a proprietary cloud, (b) a computing resource made available from a third party, like EC2 (c) a proprietary or publicly available grid?
- What different computational resources should be considered for executing GPSR systems? How does this choice set interact with GPSR representation options? How do representations suit application problems? How do applications influence computational resource choices? How much knowledge of the specific performance properties of the resources will be exposed when executing on the cloud?

The second task of this phase involves creating a detailed taxonomy of functional components we could potentially make available to the cloud-based genetic programming compiler as we invent it. Some taxonomic branches are

[1]Our first implementation will take advantage of the obvious parallelization in an evolutionary algorithm.

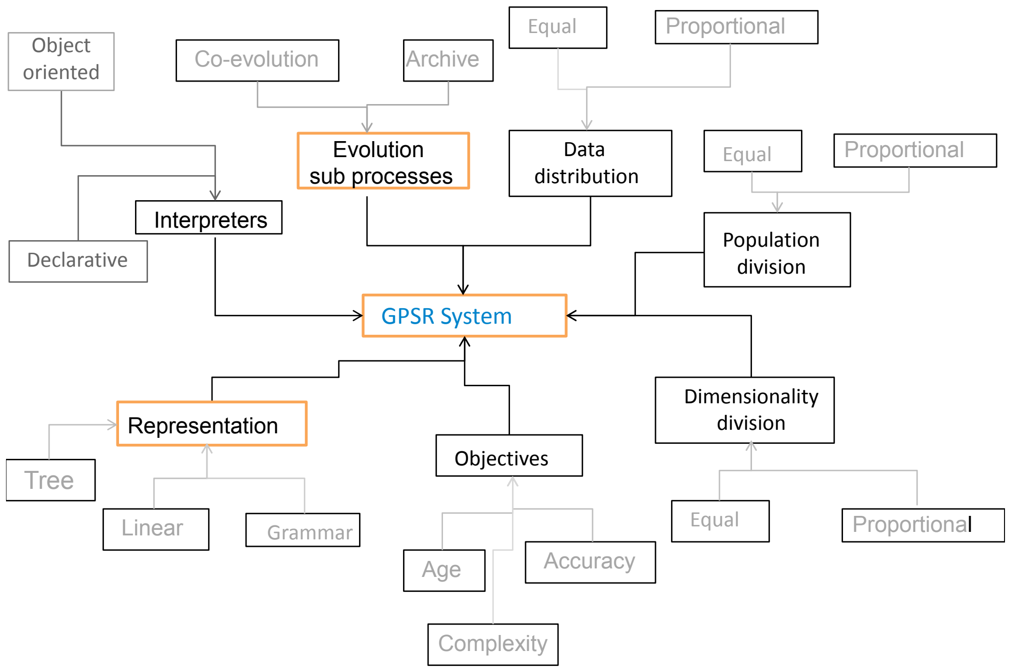


Figure 3: A preliminary taxonomy of GPSR components. A choice must be made in each taxonomic branch to compile a GPSR system. We will create a comprehensive taxonomy incorporating the innovations made by the GP community in the past two decades.

shown in Figure 3 including representation, interpreters, evolutionary sub-processes, data distribution, dimensionality distribution, population division, and objective division.

This taxonomy will allow a road map (or a decision path) to be drawn through it to represent any single instantiation of a GPSR system. As the number of branches and sub-taxonomies increase, the number of possible GPSR instantiations increases exponentially. This is key to a compiler approach. The compiler must have the option of chaining together an effective array of components (under the constraints of a rational design logic). As a third task, in Phase 1 we will start the functional definition, in detail, of the cloud-based genetic programming compiler. We will try to define it in stages.

4.2 Phase 2: A Flexibly Factored and Scaled GPSR Prototype

In Phase 2, we will define two or three realistic use cases (scenarios) for the cloud-based genetic programming compiler. A use case consists of a realistic application, its AI-level requirements and the parameters of its cloud resources. It helps define a focused, well motivated set of application specific properties and evolutionary specific properties for a flexibly factored and scaled GPSR prototype. We will then select hand-in-hand a use case and a component ensemble from the taxonomy which suit the use case. We will develop the component ensemble to run on the cloud and then synthesize them into a complete system. We will then examine if we can hand tune the components in order to maximize their efficacy to meet the application requirements on a specific resource set. This will allow us to determine in what ways the hand customized cloud system is superior to the basic GP system we developed in the first phase.

4.3 Phase 3: Automatic Compilation of a Flexibly Factors and Scaled GPSR System

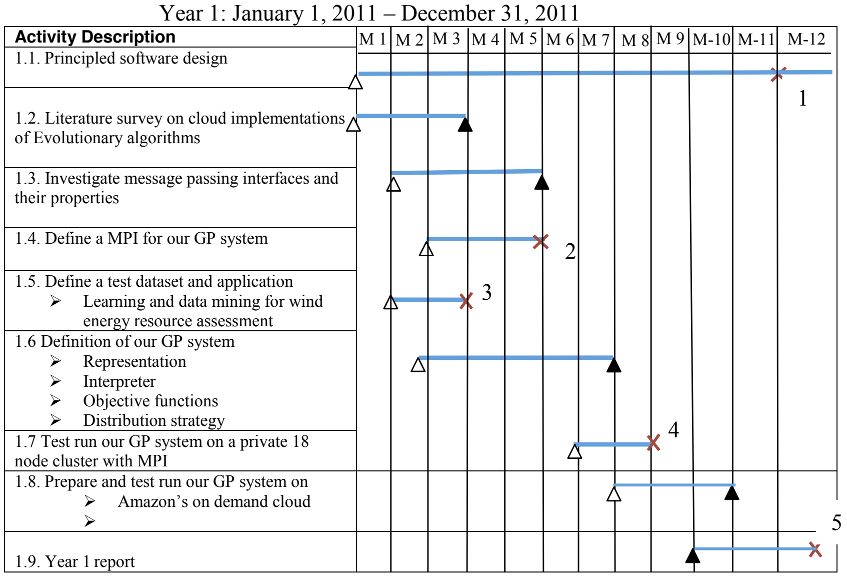
In Phase 3 our goal is to move from a hand design prototype to a rudimentary GPSR system compiler that will allow automatic compilation of GPSR systems. The compiler will start with a library of components identified from the taxonomy. Some of these will have been used in the flexibly factored and scaled GPSR prototype. To the compiler these components are choices from which it must select an ensemble to synthesize a GPSR system. The compiler (functionally specified in Phase 1) will be informed by application requirements and available compute resources. The compiler will attempt to address which salient features of an application and the architecture guide its choices. As a first step the compiler will likely solely use conventional software logic. The human in the loop will tune the system for performance after compilation.

4.4 Phase 4: Autotuning Genetic Programming Cloud Compiler

The compiler version of Phase 3 has a limitation: not only is it too complicated to specify exhaustively the compiler logic for all possible paths that can be taken to synthesize a GPSR system, there exists no adequate logic to comprehensively determine always appropriate synthesis choices. The best choice cannot be determined at compile time without candidate systems being tested and evaluated. Candidate performance will depend upon specifics of the underlying cloud architecture. In Phase 4 we will focus on mitigating this limitation. We will develop an auto tuning capability within the compiler. This moves the task of providing an *efficient* GPSR system completely to a compiler. During the compilation phase, the compiler will generate multiple component ensembles and explore their performance on the target computing resource set. Based upon this testing, it will generate one or more systems with performance envelopes.

4.5 Technical Research Performance Schedule

Year 1: January 1, 2011 – December 31, 2011



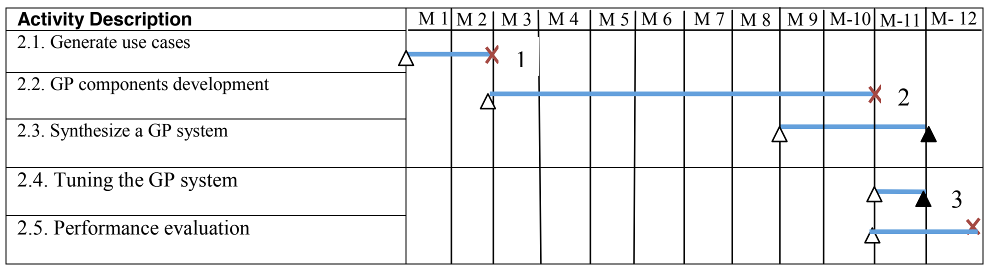
Activity Description	M 1	M 2	M 3	M 4	M 5	M 6	M 7	M 8	M 9	M-10	M-11	M-12
1.1. Principled software design	△										✕ 1	
1.2. Literature survey on cloud implementations of Evolutionary algorithms	△		▲									
1.3. Investigate message passing interfaces and their properties		△			▲							
1.4. Define a MPI for our GP system			△		✕ 2							
1.5. Define a test dataset and application ➢ Learning and data mining for wind energy resource assessment		△	✕ 3									
1.6 Definition of our GP system ➢ Representation ➢ Interpreter ➢ Objective functions ➢ Distribution strategy		△					▲					
1.7 Test run our GP system on a private 18 node cluster with MPI						△		✕ 4				
1.8. Prepare and test run our GP system on ➢ Amazon's on demand cloud ➢								△		▲		
1.9. Year 1 report										▲		✕ 5

△: Start, ✕: Milestone (MS), ▲: Breakpoint

Milestones and Deliverables
1. Software checkout and review (M-11)
2. Formal definition of MPI for our system (M-5)
3. Creation of a test dataset for learning (M-3)
4. Test run of our GP system (M-8)
5. Year 1 final report and demo (M-12)

Project Meetings
1. M-1 (January, 2011), Telecon
   Topic: Project kick start
2. M-3 (March, 2011)- MIT, Cambridge
   Topic: Discussion about a variety of existing cloud implementations
3. M-7 (July, 2011) – MIT, Cambridge
   Topic: GP system definition and a test run
4. M-10 (October, 2011) – MIT Cambridge
   Topic: Discussion about GP system on a cloud
5. M-12 (December, 2011)- GE-GRC, Niskayuna, NY
   Topic: Software checkout, review, annual report for Year 1

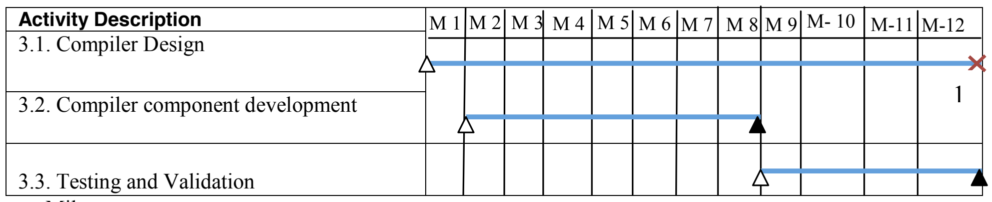
Phase 2: January 1, 2012 – December 31, 2012



Milestones
1. Use scenarios completion (M-2)
2. Component beta version (M-10)
3. Prototype beta version (M-12)

Deliverables:
Open source release of software
Project Meetings: Biweekly telecon and Monthly meetings

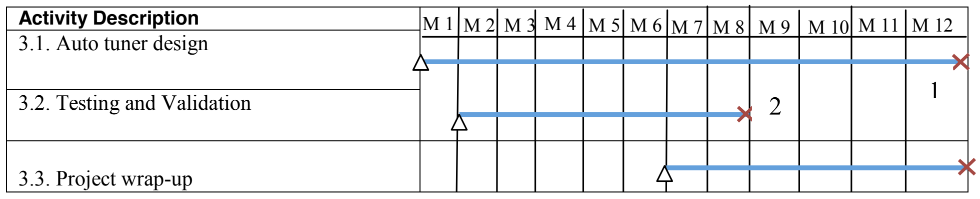
Phase 3: January 1, 2013 – December 31, 2013



Milestones
1. Release of beta version (M-11)

Deliverables: Open source release of software
Project Meetings: Same as above

Phase 4: January 1, 2014 – December 31, 2014



Milestones:
1. Auto tuner design prototype
2. Test and validation completed

Deliverables: Open source release of software
Project Meetings: Same as Phase 3

References

[1] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” vol. 51, pp. 107 –113, 2008.

[2] Spring Source, “Rabbitmq: Messaging that just works,” *http://www.rabbitmq.com/*, 2010.