Flower: A Data Analytics Flow Elasticity Manager *

Alireza Khoshkbarforoushha [1,2] , Rajiv Ranjan [3], Qing Wang [1], Carsten Friedrich [2]
[1] The Australian National University, Canberra, Australia
[2] Data61 CSIRO, Canberra, Australia
[3] Newcastle University, UK
[1] qing.wang@anu.edu.au, [2] first.last@data61.csiro.au, [3] raj.ranjan@ncl.ac.uk

ABSTRACT

A data analytics flow typically operates on three layers: ingestion, analytics, and storage, each of which is provided by a data-intensive system. These systems are often available as cloud managed services, enabling the users to have pain-free deployment of data analytics flow applications such as click-stream analytics. Despite straightforward orchestration, elasticity management of the flows is challenging. This is due to: a) heterogeneity of workloads and diversity of cloud resources such as queue partitions, compute servers and NoSQL throughputs capacity, b) workload dependencies between the layers, and c) different performance behaviours and resource consumption patterns.

In this demonstration, we present Flower, a holistic elasticity management system that exploits advanced optimization and control theory techniques to manage elasticity of complex data analytics flows on clouds. Flower analyzes statistics and data collected from different data-intensive systems to provide the user with a suite of rich functionalities, including: workload dependency analysis, optimal resource share analysis, dynamic resource provisioning, and cross-platform monitoring. We will showcase various features of Flower using a real-world data analytics flow. We will allow the audience to explore Flower by visually defining and configuring a data analytics flow elasticity manager and get hands-on experience with integrated data analytics flow management.

1. INTRODUCTION

Recent analysis of cloud providers' service portfolios [10] shows that the number of data processing platforms offered as cloud managed services has surged. This is because they are well appreciated by the users, releasing them from the hassle of platforms or cluster setup and maintenance. For example, Fig. 1 depicts a click-stream data analytics flow in which Amazon Kinesis [4] is used for managing the ingestion of streaming data at scale and Apache Storm [6] deployed on EC2 processes streaming data and persists the aggregated results in DynamoDB [3].

Figure 1: A click-stream data analytics flow.

Despite straightforward orchestration, elasticity management of an established flow is highly challenging. This is due to three unique characteristics of cloud-hosted data analytics flows. First, data analytics flow applications have heterogeneous workloads, in which different platforms and workloads are dependent on each other. For example, Fig. 2 shows how the workload dynamics in the ingestion layer is strongly correlated with the analytics layer. To provide smooth elasticity management, these workload dependencies need to be detected and considered in resource allocations.

Second, data analytics flow applications often deal with immense data volume which, together with uncertain velocity of data streams, leads to changing resource consumption patterns. This mandates an elasticity technique that could sustain workload fluctuations time efficiently, meaning that resources should be acquired and released as soon as required. However, almost all the auto-scaling systems offered by cloud providers such as Amazon [1] use simple rule-based techniques that quickly trigger in response to predefined threshold violations. Although these rules can identify fatal conditions, they often fail to adapt to unplanned or unforeseen changes in demand. Moreover, they entail considerable manual efforts in tuning each system individually and specifying rules based on available resources, which require solid expertise and planning.

Third, a data analytics flow is often deployed on heterogeneous cloud resources, each of which exhibits different performance behaviours and pricing schemes. Resource allocation thus needs to cater for diverse resource requirements and their cost dimensions to meet the users' Service Level Objectives (SLOs). Existing solutions [11, 12, 13] lack a holistic approach for resource management of big data analytics workloads. Instead, they focus either on a specific resource type such as Virtual Machines (VMs) or particular workload like Hadoop. Nevertheless, it is noted in [15] that the ability to scale down both web servers and cache tier leads to 65% saving of the peak operational cost, compared to 45% if we only consider resizing the web tier.

*Flower is an open-source tool and can be downloaded at https://github.com/Alireza-/Flower

Proceedings of the VLDB Endowment, Vol. 10, No. 12

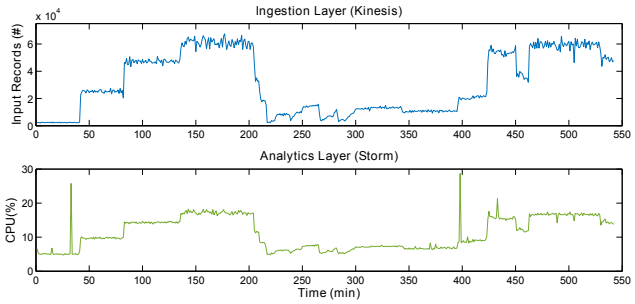

Figure 2: The data arrival rate at the ingestion layer (Kinesis in Fig. 1) is strongly correlated (coefficient = 0.95) with the CPU load at the analytics layer (Storm).

In response to these challenges, we introduce Flower, a new elasticity system that is designed for holistic resource management and performance monitoring of cloud-hosted data analytics flows. Flower's goal is to provide a high level easy-to-use system for the admins and DevOps engineers in order to fulfil the following tasks:

- **Workload Dependency Analysis.** To analyse the dependencies between workloads, Flower applies statistical regression models to workload logs to quantitatively explain relationships between, for example, resource amount in the ingestion layer (e.g. No. of Shards) and resource amount in the analysics layer (e.g. CPU usage).
- **Resource Share Analysis.** To determine how to best allocate budget to various types of resources across a data analytics flow, Flower uses a *resource share analyzer* module which uses optimization theory to resolve maximum share of different resources for each layer in a data analytics flow.
- **Resource Provisioning.** To enable accurate yet timely resource provisioning, Flower uses advanced control theory to automatically reason about resource resizing actions. Our control system is equipped with the novel feature of having memory of recent controller decisions which leads to rapid elasticity of the flow in response to workload dynamics.
- **Cross-Platform Monitoring.** Flower also provides a cross-platform monitoring module that allows the admins to observe and control all performance measures pertaining to multiple data processing systems all in one place.

In our demonstration, we will set up a data analytics flow on cloud, and then ask our audience to visually build and configure a management layer on top of the data analytics flow and observe its performance live. The audience will then be able to witness how Flower dynamically responds to workload changes.

2. FLOWER ARCHITECTURE

Flower's architecture consists of several major components, as depicted in Figure 3. In a nutshell, the workflow of the system is described as follows. First, dependencies between workloads' resource usage measures such as Kinesis Shard utilization, Storm cluster CPU usage, and DynamoDB read/write units are analyzed. To this end, we

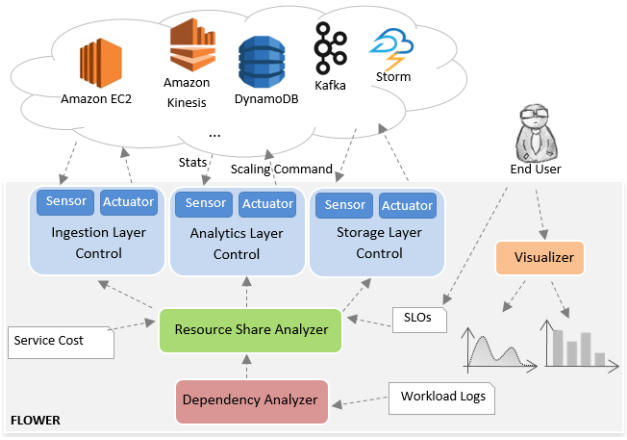

Figure 3: Flower architecture.

apply regression techniques to estimate the relationships among variables. The dependency information along with the cloud services costs and the user's SLO constitute the required inputs for the generation of provisioning plan space.

The resource share analyzer module is then invoked to determine the maximum resource shares of each layer. The resource shares can be determined with respect to arbitrary time windows. Once the *upper bound* resource shares for each layer are identified, an adaptive controller at each of the three layers automatically adjusts resource allocations of that layer.

The controllers are regulated based on a number of parameters, including monitored resource utilization value, desired resource utilization value, and history of the controller's decisions. In other words, the controllers continuously provision resources to serve the incoming records or input data in order to keep resource utilization of each layer within the specified *desired value*. For this purpose, the controllers are equipped with two key components: *sensor* and *actuator*.

The sensor module is responsible for providing resource usage stats as per the specified monitoring window. The actuator is capable of executing the controllers' commands, such as adding or removing VMs and increasing or decreasing number of Shards.

3. FLOWER COMPONENTS

3.1 Workload Dependency Analysis

To analyse workload dependencies, we use regression as a simple yet effective machine learning technique that has a sound statistical basis. Flower uses linear regression model to estimate relationships between resources in different layers. In mathematical terms, dependency between a resource r of the layer L_1 and a resource r of the layer L_2 is defined as:

$$r^{(L_1)} = \beta_0 + \beta_1 r^{(L_2)} + \epsilon \quad (1)$$

where $L_1 \neq L_2$ and they belong to either Ingestion (I), Analytics (A), or Storage (S) layers, i.e. $L_1, L_2 \in \{I, A, S\}$. β_0 is the y-intercept, β_1 is the slope, and ϵ is the error. For example, in Fig. 2 the dependency between the ingestion and the analytics layers is formulated as:

$$CPU \simeq 0.0002 * WriteCapacity + 4.8 \quad (2)$$

Evidently, Eq. 2 provides a sound starting point for the user to reason out, for example, how much CPU we require in the analytics layer to support the maximum write capacity of a Kinesis Shard in the ingestion layer - given each Shard supports up to 1,000 records/second for writes. It is worth mentioning that not all the layers are dependent on each other where, for example, we witnessed no correlation between the write capacity in Kinesis and write capacity in DynamoDB for the click-stream analytics flow application.

3.2 Resource Share Analysis

Given the budget and estimated dependencies between workloads, what would be the maximum share of resources for each layer in a data analytics flow? To address such a question, Flower provides an optimization module that is able to find an optimal solution with respect to the users' SLOs. To this end, it formulates a multi-objective function as follows:

$$\max(r_t^{(I)}, r_t^{(A)}, r_t^{(S)}) \tag{3}$$

subject to:

$$\sum_d r_t^{(I)} * c_d + \sum_d r_t^{(A)} * c_d + \sum_d r_t^{(S)} * c_d \leq Bud_t \tag{4}$$

$$r_t^{(L1)} = \beta_0 + \beta_1 r_t^{(L2)} + \epsilon \tag{5}$$

where $r_t^{(I)}, r_t^{(A)}, r_t^{(S)} \in \mathbb{R}_+$, and variables like $r_t^{(I)}$ represents the resource amount in the layer I at time t. c_d refers to the cost dimension d of the resource. The resource shares of the different layers are subject to the following constraints:

(4) *Budget Constraint*: at time t the sum of costs concerned with different cloud resources across all the layers must be within the specified budget Bud_t.

(5) *Dependency Constraints*: dependency between the layers which is learned via the regression techniques as discussed in Section 3.1.

In multi-objective optimization, there does not typically exist a solution that minimizes or maximizes all objective functions simultaneously. Thus, attention is paid to Pareto optimal solutions, i.e. those that cannot be improved in any of the objectives without degrading at least one of the other objectives. Flower uses NSGA-II algorithm [8] to efficiently search the provisioning plan space.

Consider the click-stream data analytics flow in Fig. 1 and the following *assumptive* dependency constraints as well: $5 * r_{t1}^{(A)} \geq r_{t1}^{(I)}$, $2 * r_{t1}^{(A)} \leq r_{t1}^{(I)}$, $2 * r_{t1}^{(I)} \leq r_{t1}^{(S)}$, where $r_{t1}^{(I)}$, $r_{t1}^{(A)}$, and $r_{t1}^{(S)}$ respectively refer to the number of Shards in ingestion, VMs in analytics, and Write capacity units in the storage layer at time $t1$. Given these constraints, the algorithm finds six Pareto optimal solutions, each representing the resource shares of Kinesis, Storm, and DynamoDB simultaneously, as shown in Fig. 4. In the end, one solution which is best suited to the problem in practice must be identified either manually by the user or randomly by the system.

3.3 Resource Provisioning

Flower has multiple built-in adaptive controllers tailored to the data ingestion, analytics, and the storage layers. This is due to the design principles [9] mandated by practical

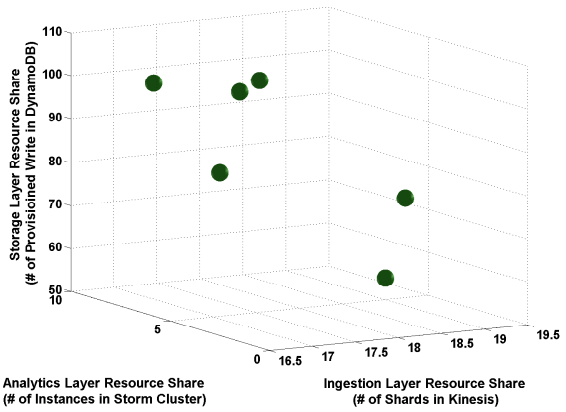

Figure 4: Pareto optimal solutions

requirements of each layer of a data analytics flow. Our adaptive controllers are defined as follows:

$$u_{k+1} = u_k + l_{k+1}(y_k - y_r), \tag{6}$$

Here, u_k and u_{k+1} are the current and new actuator values. y_k and y_r are respectively the current and desired reference sensor measurement. The controller gain l_{k+1} is adaptively updated according to the following update law:

$$l_{k+1} = \begin{cases} l_k + \gamma(y_k - y_r), & \text{if } l_{\min} \leq l_k + \gamma(y_k - y_r) \leq l_{\max} \\ l_{\min}, & \text{if } l_k + \gamma(y_k - y_r) < l_{\min} \\ l_{\max}, & \text{if } l_k + \gamma(y_k - y_r) > l_{\max} \end{cases} \tag{7}$$

Here, l_k is the controller gain at the time k, $l_{\min} > 0$ and $l_{\max} > 0$ are the lower bound and the upper bound of the controller gain, respectively, and $\gamma > 0$ is a controller parameter.

Flower's sensor module periodically collects live data from multiple sources such as CloudWatch [2] and inserts them into the actuator module. The controllers regulate the actuator value from the previous time step proportionally to the deviation between the current and desired values of the sensor variable in the current time step.

Our control system, unlike the existing solutions [12, 14], has the feature of updating the gain parameters in multi-stages and keeping the history of the previously computed control gains for rapid elasticity. Our experiments in [9] have shown that our control system outperforms the state of the art fixed-gain [12] and quasi-adaptive [14] counterparts. We have also provided a rigorous stability analysis of the resulting controllers in [9].

3.4 Cross-Platform Monitoring

Many cluster monitoring tools such as Ganglia [5] are available to assist administrators. However, they fail to provide a holistic view of performance measure across a data analytics flow. One is required to check out different systems and user interfaces in order to track any possible performance failures or slowdowns. For example, monitoring an analytics flow application built upon Storm and DynamoDB systems requires to track performance statistics in two separate user interfaces.

To tackle this issue, Flower introduces a module called *all-in-one-place visualizer*, which allows users to visually define

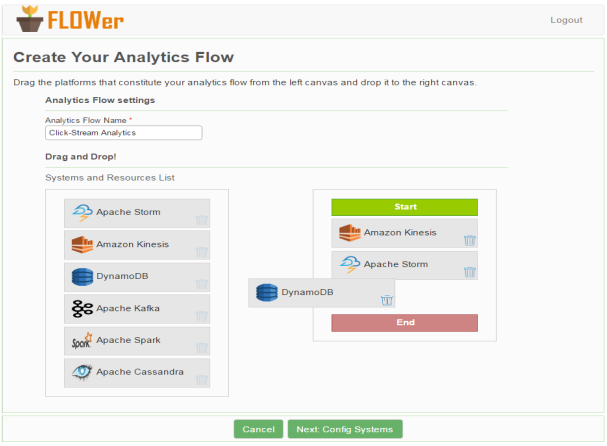

Figure 5: Flower's flow builder interface

a monitoring layer on top of multiple systems. The module calls the APIs of the systems, such as CloudWatch and Storm, and consolidates diverse performance measures in an integrated user interface:

4. DEMONSTRATION

The demonstration will give a walk-through over the key features of Flower. To this end, a demo infrastructure will be set up on Amazon cloud with three systems: (a) Kinesis, (b) Storm, and (c) DynamoDB. These systems are required to deploy the test data analytics flow - click-stream analaytics as shown in Fig. 1, which is based on the Amazon's reference architecture for click-stream analytics [7]. Once the flow is deployed, we will use a random multi-threaded click stream generator deployed on several EC2 instances to emulate the real website traffics.

We will then ask the audience to follow the steps below in order to create and run a flow elasticity manager:

1. **Flow Builder:** The attendee first will use Flower's Flow Builder to drag and drop multiple platforms and create a data analytics flow via its graphical user interface as shown in Fig. 5.
2. **Flow Configuration Wizard:** Then, the user will follow a wizard to configure the controllers with information such as resource name (e.g. table name in DynamoDB), desired reference value, and monitoring period for the selected systems in Step 1.
3. **Controller Performance Monitor**: Once configuration is completed, the user will then be able to run the service. Flower will accordingly launch visualizations as shown in Fig. 6. The attendees will then observe how different controllers change the cloud services capacities dynamically and the resulting performance. They will also be able to adjust parameters of the controllers, such as elasticity speed, monitoring period, or even their internal settings and compare their impacts on SLOs.

In addition to above demonstration, we will also ask the attendees to create a cross-platform visualization dashboard using similar steps. This will allow them to experience live monitoring of multiple systems all in one go.

5. REFERENCES

[1] Amazon auto scaling. `https://aws.amazon.com/autoscaling/`.
[2] Amazon cloudwatch. `https://aws.amazon.com/cloudwatch/`.
[3] Amazon dynamodb. `https://aws.amazon.com/dynamodb`.
[4] Amazon kinesis. `http://aws.amazon.com/kinesis`.
[5] Ganglia monitoring system `http://ganglia.sourceforge.net/`.
[6] Apache storm. `http://storm.apache.org/`.
[7] R. Bhartia. Amazon kinesis and apache storm: Building a real-time sliding-window dashboard over streaming data. Technical report, Amazon Web Services, October 2014.
[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *TEVC*, 6(2):182–197, 2002.
[9] A. Khoshkbarforoushha, A. Khosravian, and R. Ranjan. Elasticity management of Streaming Data Analytics Flows on clouds, *Journal of Computer System Sciences, 2016.*
[10] A. Khoshkbarforoushha, M. Wang, R. Ranjan, L. Wang, L. Alem, S. U. Khan, and B. Benatallah. Dimensions for evaluating cloud resource orchestration frameworks. *Computer*, 49(2):24–33, 2016.
[11] I. Konstantinou, E. Angelou, D. Tsoumakos, C. Boumpouka, N. Koziris, and S. Sioutas. TIRAMOLA: elastic NoSQL provisioning through a cloud management platform. In *SIGMOD*, pages 725–728. ACM, 2012.
[12] H. C. Lim, S. Babu, and J. S. Chase. Automated control for elastic storage. In *ICAC*, pages 1–10. ACM, 2010.
[13] J. Ortiz, B. Lee, and M. Balazinska. Perfenforce demonstration: Data analytics with performance guarantees. SIGMOD, 2016.
[14] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 289–302. 2007.
[15] T. Zhu, A. Gandhi, M. Harchol-Balter, and M. A. Kozuch. Saving cash by using less cache. In *HotCloud*, 2012.

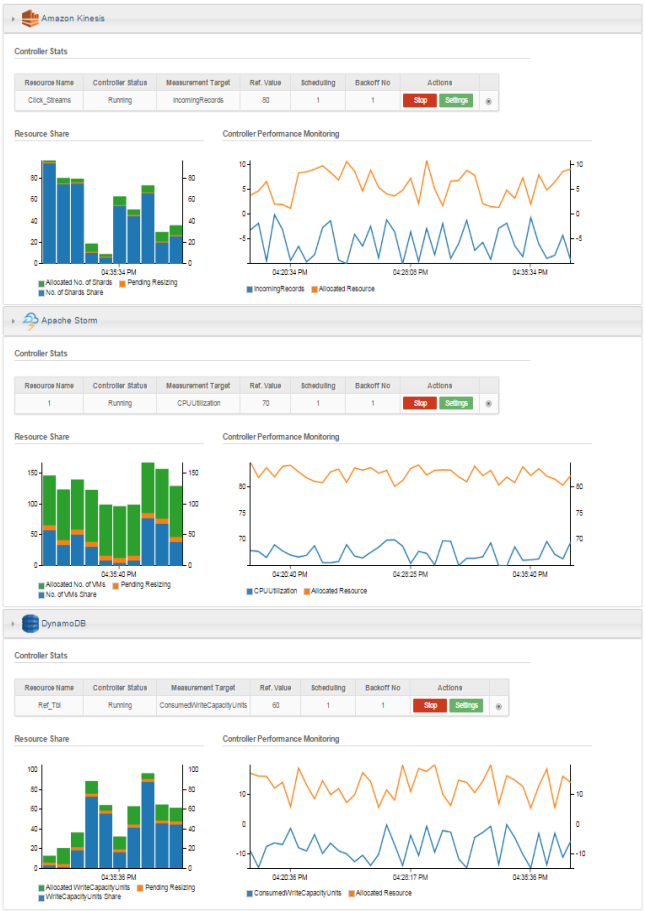

Figure 6: Elasticity control and monitoring interface