# Orbit: Efficient Processing of Iterations

Douglas Ericson M. de Oliveira, Fabio Porto

Extreme Data Lab (DEXL), Laboratório Nacional de Computação Científica
ericson@lncc.br, fporto@lncc.br

**Abstract.** Various scientific applications compute iterations on a huge set of input data. Examples include parameter sweep, which processes the same model through hundreds or thousands of input data, scientific visualization and numeric methods to solve hyperbolic equations. The state of the art for executing such applications is to model them as scientific workflows and to run them in HPC infrastructure. The execution model strives to combine pipeline parallelism, among activities of the workflow, with intra-workflow parallelism over data partitions, both of them subjected to workflow characteristics. In such scenario, the scientific workflow execution model can be approximated to that of parallel query processing. In fact, database groups at LNCC and COPPE have implemented parallel workflow engines, QEF and Chiron, under this assumption. In order to full integrate iterations within a data processing execution model, an extension is required to treat the loop control operator. In this paper we investigate the problem of efficiently executing scientific workflows that include iterations. We introduce Orbit which is a generic operator to manage the data flow in an iterative procedure. An execution model centered into Orbit is proposed including a centralized and parallelized modes. Additionally, two new execution strategies are investigated: first-tuple-first, first-iteration-first. We have obtained initial results that evaluate the different execution strategies in both centralized and parallel modes.

Categories and Subject Descriptors: H.Information Systems [**H.m. Miscellaneous**]: Databases

General Terms: Scientific Workflows, Execution Model

Keywords: Orbit, Control Operator, Query Processing, iteration

## 1. INTRODUCTION

The adoption of scientific workflow model to implement large scale science experiments and data analysis applications is the state of the art in eScience. Systems designed to process scientific workflows are known as workflow engines and a handful of them are available for the community to use. Among the different workflow engines, a particular architecture that is of interest to this paper evaluates huge volume of data in parallel computers. Applications such as parameter sweep, wherein a scientific model is evaluated against a huge parameter value space; and scientific visualization, in which the results of numeric simulations are browsed in time-space bringing a realistic view of a phenomenon, are examples of scientific applications that iterate over a number of steps or over input instances.

In previous work [Porto et al. 2007; Ogasawara et al. 2011], we have highlighted the similarities between the execution model of scientific workflows and that of query processing. Under this perspective, a workflow model is a generalization of data processing models of which query processing is a simple case. In particular, the execution of queries is governed by query execution plans represented as trees, left deep or bushy [Ozsu and P.Valduriez 2011]. This restricted topology is extended in workflow models. The reason why this analogy is relevant is that it may lead to the application of cost based query optimization strategies to workflow models, in addition to the reuse of query processor machinery to run scientific workflow. That is exactly the path that the QEF system [Porto et al. 2007] has followed. Thus, it is instructive to identify workflow models whose execution behavior would not be supported by workflow engines and would limit the applicability of query optimization techniques.

In this context, the applications referred above are examples of such category. Indeed, the work on query optimization has developed on the relational algebra [Codd 1970] whose semantics does not include program flow control structures, such as iterations. A line of work explored the execution of transitive closure [Sippu and Soisalon-Soininen 1988] over tuples of a relation that basically consider a special case of iteration, one that appears in self-joins. Note that we are interested in more general iterations, in which tuples may iterate over a fragment of the workflow.

In order to illustrate a typical application, consider the virtual particle trajectory (VPT) visualization application. At the Hemolab Laboratory [1], a computational model of the cardio-vascular system simulates the flow of blood through arteries. The VPT application receives a mesh, geometrically representing parts of a human artery, and, for each point in the mesh, the velocity of the flow at each time instant. Moreover, VPT receives a set virtual particle representations of the blood with a initial position in space-time. The workflow includes three operations: a spatial join, that matches each particle position within a mesh geometrical figure; a join in space-time that for a given set of points look for their velocity in a time instant; and a trajectory computing program that calculates the next position for the virtual particle. Thus, this fragment of the workflow must be iterated the number of times corresponding to the time-range of the simulation. The iterative evaluation of a fragment of a workflow introduces a cycle into graph representation of the workflow.

In this context, this paper contributes to the execution model of scientific workflows with a data processing operator, named *Orbit*, that implements data iteration through a fragment of a workflow. Orbit introduces cycles into workflow execution while retaining the generic behavior of query processing operators bringing forth the integration of query processing strategies with scientific workflow execution models. An experiment has been conducted comparing two execution models: $First\_Tuple\_First$ and $First\_Iteration\_First$, and two distribution models: Master and Remote.

## 2. RELATED WORK AND CONCEPTS

Processing huge amount of data is currently a hot topic, sometimes encapsulated around the buzzword *Big Data*. An important initiative in these lines is to extend databases with user defined functions (UDF) [Stonebraker and Rowe 1986], which has been adopted within the Sloan Digital Sky Survey project [2], or adding full execution environments, such as [Liae et al. 2008], which deals with iteration within the package. Other approaches, such as Oracle pl/sql add full fledged programming language to be used in stored-procedures and UDFs. None of these initiatives deal with the introduction of iteration such that it can be taken into consideration during query optimization.

Another area of research with results over time is related to the computation of the transitive closure operation on relations, with special attention given by the expression of recursion in Datalog [Abiteboul et al. 1994]. In both cases, the operation is evaluated as a self-join. A more recent initiative Haloop [Bu et al. 2010] implements iterations as a sequence of joins with aggregates using the Hadoop system [Had 2013], focusing in collocating at the same node operations that communicate data.

In the workflow scenario, such approaches need to be extended to cope with data iteration through a fragment of the workflow. Orbit has been inspired by the n-ary Eddy operator [Avnur and Hellerstein 2000]. Eddy has been proposed to introduce adaptivity to a query execution plan. In fact, the operator achieves an iterative behavior almost by chance, while breaking the full ordering of operators in a query execution plan. Thus, in the perspective of this paper, Eddy could be considered to implement two different behaviors:iteration and adaptivity. In this line, the present work clarifies this issue in a way that Orbit could be used in association with another operator to deal with adaptation. More importantly, Orbit implements iteration irrespectively of an adaptive or fixed execution strategy.

[1]http://macc.lncc.br
[2]http://www.sdss.org

### 2.1 QEF

Orbit has been conceived as an operator following the known *iterator* interface [Graefe 1990], thus it can be integrated in any modern query processor. Our implementation, nevertheless, was done using the QEF (*Query Engine Framework*).QEF is a data processing engine designed as an extension based on query processor technology. In QEF a data processing application is implemented by extending three main structures: Data Unit, Algebraic Operators and Control Operators.

A QEF workflow defines a DAG, where nodes correspond to operators and directed edges represent the flow of data from a producer to a consumer operator. In this work, we have extended QEF model by supporting cyclic workflows with the introduction of the Orbit control operator.

## 3. THE ORBIT OPERATOR

In this section, we introduce the Orbit operator. We initially give some basic definitions used during the operator specification.

*Definition* 3.1. A workflow model is a partial ordered set of operations. Each operation consumes and produces a Data Unit. An operator $op_j$ succeeds $op_i$, $op_j \succ op_i$, in a workflow model $w$, if there is a data item $d$ that is produced by $op_i$ and consumed by $op_j$, directly or indirectly.

*Definition* 3.2. A Fragment of Workflow Model $\phi$ is a subset of operators in a workflow model $w$, such that for each pair of operators $op_i$ and $op_j$, either $op_i \succ op_j$ or $op_j \succ op_i$ in $\phi$. A fragment of workflow is limited by a bottom and top operators. A bottom operator in a fragment of workflow $\phi$, $op_b \in \phi$, is such that $op_b$ directly $\succ op_l$ and $op_l \notin \phi$. The bottom operator directly succeeds the top operator, in a fragment of a workflow.

*Definition* 3.3. Cyclic fragment of a workflow (CFW) - is such that exist two operators $op_i, op_j \in \phi$, with $op_j \succ op_i$ and $op_i \succ op_j$.

Given a cyclic fragment of a workflow $\phi$, the Orbit operator is placed in $\phi$ such that the bottom operator in $\phi$ directly $\succ$ Orbit, and the latter directly $\succ$ top.

### 3.1 Semantics

Orbit is a quaternary control operator defined as *Orbit(Tuple inpipeline, Tuple outpipeline, Tuple inorbit, Tuple outorbit)*, such that *inpipeline* and *outpipeline* correspond to the tuples coming from *producer* and leaving to *consumer* operators, respectively (see below). Conversely, within the CFW, the *inorbit* and *outorbit* correspond to tuples feeding, and returning from, the internal loop, respectively. The operator is placed in a workflow to implement the cyclic behavior in a fragment of a workflow model. A cyclic workflow model enables the iterative evaluation of tuples through the operators taking part in the workflow fragment.

Figure 1 presents Orbit components, also showing the relationship among its respective producers/consumers. Each one of these components, together with their corresponding responsibilities in the cyclic execution model implemented with Orbit are described next.

—Producer: is responsible for providing the tuples to be iterated by the cyclic fragment of the workflow (CFW). Orbit implements the iterator operator interface [Graefe 1990] and issues *getNext* calls to the producer operator. An independent thread is charged to call this function and to store the tuples into the Orbit buffer. This thread is controlled by a semaphore that warns when a tuple can or cannot be loaded into the buffer. Therefore, a new tuple cannot be loaded until old tuples have not finished their execution (i.e. either finished their iterations or are eliminated by one operator within the fragment), otherwise the thread is blocked. In this paper, such thread is called a feeding thread. The buffer stores all tuples generated by the *Producer* operator that have not yet been consumed by the *Consumer* operator and that are not being processed by the fragment of the workflow;

—Consumer: is an operator $op_l \notin \phi$, such that $op_l$ directly $\succ$ Orbit. It obtains tuples that have achieved the desired number of iterations.

—CFW Producer Operator: corresponds to the top operator of a CFW. It is succeeded by the Orbit operator. Orbit evaluates whether the tuple produced by the CFW Producer Operator is subject to a new iteration, in which case it is stored in the buffer that feeds the CFW Consumer Operator;

—CFW Consumer Operator: conversely, it corresponds to the bottom operator in a CFW. It consumes tuples that are in the Orbit buffer, according to the Iterator model. It is worth observing that the consumed tuples are indeed eliminated from the buffer, but they may eventually be returned to it during Orbit evaluation;

—Functions: as already described, Orbit main goal is to continuously evaluate a tuple until it reaches a certain requirement that eliminates it from the cycle. Orbit may implement any end of iteration criteria using boolean functions, which are specific to each application. For example, considering the TCP application, each processed tuple is sent to the Orbit operator that increments the number of iterations performed by the tuple and determines whether it will be re-evaluated by the CFW operators, or if it will leave the execution environment. In the latter case, the tuple is consumed by the Consumer operator, which is responsible for performing the final procedures according to this application. Besides this tuple controlling task, Orbit also includes other two important functionalities: the first, called iteration, is a property assigned to each tuple when it is sent to the buffer for the first time, before its execution by the CFW operator. This property aims at identifying the number of iterations performed by each tuple, and its initial value is set to zero. The second is to make a copy of each tuple before sending it to the Consumer operator, in case Orbit decides the tuple must be re-executed by CFW.

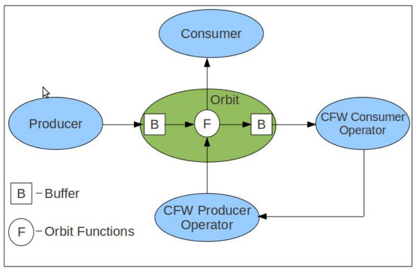

Fig. 1. Orbit Architecture and Relationships among its Operators.

## 4. PARALLELISM USING ORBIT

Orbit can be executed using three different approaches: (1) Centralized,(2) Parallel with Orbit as a master node and (3) Parallel with Orbit on each remote node. In the (1) all operators in the fragment of the workflow run in the same machine. The cyclic behavior is implemented by introducing the Orbit operator into the workflow model defining the CFW.

In (2) there is a master node and set of remote nodes. The remote nodes are used to enable the parallel execution of a fragment of the workflow. In particular, each remote node may run an instance of the CFW enabling the fragment parallel execution. When Orbit is run in the master node, the whole CFW is placed in the remote nodes. Data consumed by the bottom operator is sent in blocks through the network. Once the tuples in a block have been processed by the CFW they are packed back into a block and sent to the Orbit operator.

In (3) the complete CFW, including Orbit are placed at the remote nodes. The *Producer* operator communicates with the CFW through the Control operators dealing with block transfer. Comparing to the *Orbit in the master* model, in this model, Orbit loses its environment adaptive functions described previously. The execution behavior on each remote node follows that of the Centralized model. A great advantage of this model is in reducing the number of block transfers through the network.

## 5. EXPERIMENTS

In this section the proposed execution methods will be evaluated. Experimental tests have been carried out within the purpose of evaluating the Orbit operator performance in iterative applications. In this context, we chose the computational technique over particle's trajectory (VPT) to elaborate these tests, in which 1000 virtual particles were processed over 10 iterations. Experiments were performed within a set of seventeen nodes (sixteen nodes dedicated for execution and a single one as a master) and each machine was composed of 2 Quad Core Intel Xeon E5520 @ 2.27GHz processors, 12GB memory, Ubuntu OS 10.04 and 1 TB hard disk.

For each Orbit execution model three implementation types were performed: First Iteration First (FIF), First Tuple First (FTF) and FREE. In all of them, the Orbit buffer responsible for the tuple storage was implemented according to a priority queue data structure. The consumption mode of the stored tuples inside the Orbit buffer determines the execution model to be applied. In the FREE model tuples are taken randomly.

In FIF tuples are synchronized according to its iterations. The purpose here is to ensure all tuples will always perform the same iteration, i.e., they will not execute the next iteration until all of them have finished executing the current one. Therefore, even if a tuple has executed its iteration faster than others, it will have to wait for the others before executing the next iteration. Hence, this execution model is characterized by not eliminating tuples until the last iteration has been executed.

The FTF model prioritizes tuples taking part of higher level iterations. Therefore, if a tuple has executed its iteration faster than others, it does not need to wait for them to execute the next iteration. Thus, it is possible to have tuples processing all their iterations, while others are still executing previous ones. The purpose here is to provide the user with a faster final answer time, ensuring that, if for any reason the tuple does not return the expected result, this occurrence can be identified during its execution and not at the end, as in FIF model.

### 5.1 Time Execution Evaluation

Figure 2 presents the execution time (seconds) for the model combination (FIF, FTF and FREE) and parallel execution mode (master - OM and remote Orbit). It is possible to notice that execution considering Orbit in each remote node is faster than its execution using Orbit only in the master node. This is due to the fact that in the OM, more time is spent with communication between the master and each remote node. At each iteration the Orbit in the master node splits its data among each remote node. Then, for each iteration, the following procedure is performed: the remote machine executes an iteration over its data part and returns the processed data to the master node. This continuous communication data flow between the master and remote nodes finishes by directly influencing the final application execution time.

When Orbit executes at each remote node, data are read, split and sent a single time by the master to each remote node, according to a parallel processing similar to MapReduce. Thus, each machine executes all the tuples received for iteration. When this process ends, data are sent back from the remote node to the master, which concludes the execution. Hence, time in this latter execution model is much shorter than the first one, when Orbit is in the master node. Nevertheless, there exists an advantage of executing Orbit in the master node. The larger scalability provided by execution in remote nodes reduces their capacity of reacting to fluctuations, and thus decreases time performance.

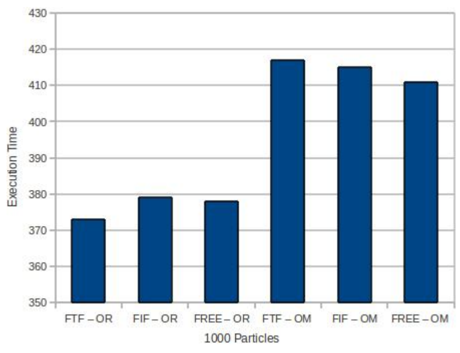

Fig. 2. Execution Time considering each Orbit Model

## 6. CONCLUSION AND FUTURE WORKS

This paper introduces Orbit an operator for the implementation of iterative behavior in data processing systems. A data processing system equipped with Orbit may produce different execution models, including centralized and distributed modes and different tuple scheduling strategies. We have implemented Orbit in QEF, a data processing system, and experimented with a real application. These first results privilege the allocation of Orbit, and the CFW, in remote nodes. More experiments may be needed to define its proper allocation in a more varied execution scenario.

## 7. ACKNOWLEDGEMENTS

This work was partially funded by CNPq, Research Productivity - RP - 2012, process:309494/2012-5. The authors would also like to express their gratitude to the ComCiDIS laboratory for allowing us to run our experiments on their cluster environment.

## REFERENCES

Hadoop http://hadoop.apache.org, Last access July 2013, 2013.

Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases:The Logical Level.* Addison-Wesley, 1994.

Avnur, R. and Hellerstein, J. M. Eddies: continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data.* SIGMOD '00. ACM, New York, NY, USA, pp. 261–272, 2000.

Bu, Y., Howe, B., Balazinska, M., and Ernst, M. D. Haloop: efficient iterative data processing on large clusters. *Proc. VLDB Endow.* 3 (1-2): 285–296, Sept., 2010.

Codd, E. A relational model for large shared data banks. *Communication of the ACM* 13 (6, 1970.

Graefe, G. Encapsulation of parallelism in the volcano query processing system. *SIGMOD Rec.* 19 (2): 102–111, 1990.

Liae, Y., Perlmanb, E., Wana, M., Yanga, Y., C. Meneveaua, R. B., S. Chena, A. S., and Eyinkd, G. A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence* 9 (2, Oct., 2008.

Ogasawara, E. S., de Oliveira, D., Valduriez, P., Dias, J., Porto, F., and Mattoso, M. An algebraic approach for data-centric scientific workflows. *PVLDB* 4 (12): 1328–1339, 2011.

Ozsu, T. and P.Valduriez. *Principles of Distributed Database Systems.* Springer, 2011.

Porto, F., Tajmouati, O., Silva, V. F. V. D., Schulze, B., and Ayres, F. V. M. Qef - supporting complex query applications. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid.* IEEE Computer Society, Washington, DC, USA, pp. 846–851, 2007.

Sippu, S. and Soisalon-Soininen, E. A generalized transitive closure for relational queries. In *PODS.* pp. 325–332, 1988.

Stonebraker, M. and Rowe, L. A. The design of postgres. *SIGMOD Rec.* 15 (2): 340–355, June, 1986.