ISSN: 2326-4411 (Print)
ISSN: 2326-442X (Online)

International Journal of Big Data

A Technical Publication of the Services Society

2015

VOL. 02 NO. 02

TASK AND DATA ALLOCATION STRATEGIES FOR BIG DATA WORKFLOWS

Mahdi Ebrahimi, Aravind Mohan, Andrey Kashlev, Shiyong Lu, Robert G. Reynolds

Wayne State University

Detroit, U.S.A.
{mebrahimi, amohan, andrey.kashlev, shiyong, robert.reynolds}@wayne.edu

ABSTRACT

The makespan of a big data workflow is the time elapsed between the start of the first task and the completion of the last task in the workflow. This time includes the delivery of the final data product to the desired location within the network. Due to the large number of inputs and intermediate outputs of a big data workflow activity, the makespan of the workflow is significantly influenced by how its tasks and datasets are allocated in a distributed computing environment. Therefore, reducing makespans of big data workflows can be achieved by incorporating a data and task allocation strategy into an execution planning phase performed by a workflow management system. This creates a pressing need for an investigation of such strategies. To address this need, this paper provides a formal definition of the makespan minimization problem for big data workflows and proposes efficient workflow execution planning strategies. In particular, two algorithms, WEP-A and WEP-B, following different strategies are proposed. WEP-A follows a phased approach to the generation of an execution plan whereas WEP-B uses an evolutionary algorithm-based optimization strategy to find a valid plan with the shortest makespan. Both of these strategies are evaluated through extensive simulation experiments by varying workflow graphs and resources in the workflow environment. The results of the experiments demonstrate that WEP-B performs better than WEP-A on a set of benchmark examples. For more complex and large workflows, the improvements due to evolutionary optimization in WEP-B are likely to be even more pronounced.

Keywords: Data Allocation, Big Data Workflows, Evolutionary Algorithm, Distributing Computing.

---

1. INTRODUCTION

Complex scientific computations are often modeled as workflows. Among other benefits, decomposing a complex application into a workflow simplifies the design effort, enables the reuse of computational modules, and allows their parallel and/or pipelined execution. The concept of workflow applications is used widely in scientific research in variety of domains such as bioinformatics, physics, astronomy, ecology and earthquake science (Lu and Zhang, 2009, Lim et al., 2010, Juve and Deelman, 2010). Given the computing, storage and networking technologies, coupled with the increased capacity to perform collaborative scientific research, there is an increased need to produce efficient scientific workflows.

A data-centric workflow management system (DWFMS) is a platform designed to support two key functions: 1) the design and specification of workflows using a visual drag and drop interface; and 2) the configuration, execution and monitoring of workflow runs. Examples of representative DWFMS systems include Taverna (Hull et al., 2006), Kepler (Ludäscher et al., 2005), VisTrails (Freire, et al., 2006), Pegasus (Deelman et al., 2005), Swift (Zhao et

al., 2007), and DATAVIEW (Kashlev and Lu, 2014, Chebotko et al., 2007). Traditionally, these systems use a directed acyclic graph (DAG) abstraction to model a workflow, where each vertex of the graph represents a workflow task, and each directed edge between two vertices depicts the dataflow between them.

In order to support the demands of big data workflows, the design interface and DAG abstraction must be expanded to explicitly model both data-centric and computational activities. The execution module must also be modified in order to include execution planning and preparation functions to decide, allocate and move data sets and tasks to their respective hosts in the target distributed computing environment. The determination of an efficient execution plan is a non-trivial exercise when the workflow involves big data sets and hundreds or thousands of complex tasks (Bharathi et. al., 2008, Deelman and Chervenak, 2008).

This paper contributes toward the enhancement of existing DWFMSs in the area of execution planning of big data workflows. Workflow execution planning is directed by the strategy adopted for the allocation of data and workflow tasks. The strategies proposed here seek to minimize the total execution time (a.k.a. makespan) of the workflow. Makespan is defined as the time that elapsed between the start of the first task and the completion of the last task in the workflow, including the delivery of final data products to a desired place. A big data workflow involves big datasets, either as inputs or intermediate outputs. Therefore, the makespan can vary greatly depending on how the tasks and datasets are allocated in the distributed computing environment. The utilization of a data and task allocation strategy that minimizes the makespan in a big data workflow management system can deliver significant benefits to users in getting their results in time.

The remainder of the paper is organized as follows. Section 2 describes the characteristics, and assumptions of the workflow execution environment needed to support such a model. An extended graphical model of big data workflows and the concept of execution plans are presented in section 3. In Section 4, a formal definition of the makespan minimization problem of big data workflows is provided. And two efficient algorithms are proposed to solve it. In Section 5, the proposed algorithms are compared with a random allocation strategy using and extensive series of simulation runs. A brief overview of related work appears in Section 6. The paper concludes in Section 7 with a summary of the key contributions of the paper.

2. BIG DATA WORKFLOW EXECUTION ENVIRONMENT

An execution environment for a big data workflow is comprised of geographically distributed autonomous sites spread around the globe. In this paper, these sites are defined as 'hosting' sites, or simply as 'hosts'. The hosts may be heterogeneous in various ways, but all have the capability to provide data hosting service and/or compute (task execution) services.

Hosts can be classified into three basic types: a data host; a task host; and a hybrid host. A "data host" provides only data hosting services. It will act on data storage and data transfer requests, but cannot be used to perform arbitrary workflow tasks. A "task host" can be used to run workflow tasks, but cannot be used for (durable) data hosting. It is able to provide enough temporary scratch storage for all input and output data sets that relate to the execution of allocated tasks. A "hybrid host" is capable of providing data hosting services as well as compute services.

For a collection of 'hosts' to serve as an execution environment for a particular big data workflow, there must be a communication capability to stream big data sets from each host to all other hosts in the collection. Furthermore, there must be an agreement that allows such streaming to take place at a predicted level of service and time. In addition 'task hosts' must be available to run workflow tasks on demand, as dictated by the workflow management system.

The notion of 'host' is a generic abstraction for packaging storage and computing capacity and does not imply a specific underlying architecture or business model. Notable examples of 'hosts' include

public clouds, private clouds, HPC data centers, and Grid nodes. Here it is assumed that the internal architectures of the 'host' implementations will provide data and computing services with predictable SLA's in terms of data transfer rates and processing speeds.

The first step in the production of a mathematical formulation of the makespan minimization problem is to characterize the properties of a "host". The three basic properties of a "host" in the model are

- *Available Storage Capacity* – This attribute is relevant to hosts of type 'data' or 'hybrid'. The 'Available Storage Capacity' refers to the capacity available for the execution of a particular workflow and does not reflect the total storage capacity of the host in question. The available storage capacity may be less than the total capacity as a result of a sharing policy or another service agreement. An elastic storage service provided by the host may allow available storage capacity to be changed. However, it is assumed in the model that such changes occur prior to and outside the execution planning of a particular workflow run. Thus, for the purpose of this formulation, available storage capacity of a 'host' is fixed and is specified using the same units, such as megabytes for each host.
- *Available Data Transfer Bandwidth* – This attribute allows the prediction of the time required to transfer a dataset from one 'host' to another. The available data transfer bandwidth may be limited by the connectivity of the host, and/or by a service level agreement.
- *Available Compute Capacity* – This attribute is applicable to hosts of type 'task' or 'hybrid' and allows the prediction of the processing time of a workflow task as well as the number of workflow tasks that can be executed in parallel on the host without impacting individual task execution times. As with storage capacity, compute capacity can be constrained by policy or service level agreements.

Note that the above three attributes are not fixed or static for a host at all times. These are considered to be established priori prior negotiations and remain unchanged during the execution planning and subsequent execution of an individual workflow.

For workflow execution planning purposes, it is assumed that all of the hosts in the workflow environment are fully reliable and available. The responsibility of ensuring these characteristics lies with individual hosts and is covered by their SLA's. The impact of the recovery and high-availability mechanisms is reflected in the expected processing times of tasks and expected data transfer rates associated with the host.

3. FORMULATION OF WORKFLOW EXECUTION PLANNING

To formalize the workflow execution planning problem, formal descriptions of the following are needed:

- Big Data Workflow Execution Environment
- Big Data Workflows
- Big Data Workflow Execution Plan

Table 1 summarizes all the symbols and notations used in the following definitions.

Definition 1: (Big Data Workflow Execution Environment) - A big data workflow execution environment Ω is defined as a 6-tuple $\Omega \equiv \{H, TYPE, ASC, ACC, DTR, CR\}$; where for all $h \in H$, $ASC(h) = 0$ if $TYPE(h) =$ 'task' and $ACC(h) = 0$ if $TYPE(h) =$ 'data'. In terms of the big data execution environment defined above, big data workflow represented as a directed acyclic multipartite graph as given below.

Definition 2: (Big Data Workflow) – A big data workflow, W, is represented as a directed acyclic graph as a 3-tuple, $W \equiv (V, E, SIZE)$, where the set of nodes, V, satisfy the following properties:

- There are two special nodes called entry and exit nodes of type 'task'.
- For each dataset referenced in the workflow there is a unique node 'x' in V where NODETYPE(x) ='data' and vice versa.

Table 1: Notations Summary

Notation	Description
Ω	Workflow Execution Environment
H	Set of hosts in **Ω.**
M	Total number of hosts in H.
h_k	k^{th} host in H; $1 \leq k \leq m$.
TYPE	A function mapping hosts to unique types; TYPE: H → ('data', 'task', 'hybrid').
ASC	A function mapping hosts to their available storage capacities. ASC: H → I, where I is the set of non-negative integers.
ACC	A function mapping hosts to their available compute capacities. ACC: H → I, where I is the set of non-negative integers.
DTR	A function mapping a pair of hosts to data transfer rates between them. DTR: H × H→ R, where R is the set of non-negative real numbers.
CR	Compute Rate Function giving compute rate for each host CR: H → R, where R is the set of non-negative real numbers.
W	Scientific Workflow Graph
V	Set of nodes in W.
NODETYPE	A function mapping nodes to node types as "Data' or 'Task'. NODETYPE: V → {'Data', 'Task'}
P	An ordered partitioning of V into disjoint sets called partitions where nodes in a partition are of identical type.
|P|	Number of partitions in **P**.
P_i	i^{th} partition in P.
E	Set of directed edges in W.
e_{xy}	Directed edge from node '*x*' to node '*y*' in V.
SIZE	A function mapping nodes to numbers. SIZE: V → I, where I is the set of non-negative integers.
pred(x)	*pred(x)*: {*y*| *y* ∈ V, where there exists an edge e_{yx} ∈ V}.
Ω	A workflow execution plan for a given workflow graph **W** and the workflow execution environment **Ω.**
Φ	Mapping of nodes in **W** to hosts **H** in **Ω.**
Γ	A scheduling function that computes the scheduled start time, the scheduled completion time, and thescheduled release time for each node in **W.** Γ: V → {Scheduled Start Time, Scheduled Completion Time, Scheduled Release Time }
$start_x$**(Γ)**	Scheduled start time for node '*x*' generated by the scheduling function Γ.
$finish_x$**(Γ)**	Scheduled completion time for node '*x*' generated by the scheduling function Γ.
$release_x$**(Γ)**	Scheduled release time for node '*x*' generated by the scheduling function Γ.
requires	A function that computes the maximum storage required for any host '*h*' of type 'data' or 'hybrid' at the same time during the execution plan ω. *requireS*: H → $_\omega$ Number of Storage Units
require	A function that computes the maximum number of tasks assigned to any host '*h*' of type 'task' or 'Hybrid' at the same time during the execution plan ω. *requireC*: H → $_\omega$ Number of Compute Tasks to be executed in parallel
restrictedHosts	List of hosts where a node '*x*' is restricted for mapping by policy. restrictedHosts: V → {set of hosts}
transfer(x, y)	Data transfer time between nodes '*x*' and '*y*' which varies depending upon the mapping of nodes to hosts.

- For each task in the workflow there is a unique node 'n' in V where NODETYPE(n) ='task' and vice versa.

The set of edges, E, satisfy the following properties:

- There is no edge $e_{xy} \in E$ where x = y, i.e., there is no edge or self-loop from a node to itself.
- There is no edge $e_{xy} \in E$, such that if 'y' is the entry node, there is no incoming edge to the entry node.
- There is no edge $e_{xy} \in E$, such that if 'x' is the exit node, there is no outgoing edge from the exit node.
- There is no edge $e_{xy} \in E$ where NODETYPE(x) = NODETYPE(y). That is, there is no edge between nodes of identical type.
- For each node y in V, there is exactly one edge $e_{xy} \in E$ if NODETYPE(y) = 'data'. That is, there is exactly one incoming edge to a datanode.
- For each node y in V, where 'y' represents a pre-existing dataset (input to the workflow), there is exactly one edge $e_{xy} \in E$ where 'x' is the entry node.

The SIZE function satisfies the following properties:

- SIZE(x) = 0, if 'x' is the entry or exit node.
- For any 'x'∈V, if NODETYPE(x) = 'data', then SIZE(x) represents the size of the corresponding dataset in units of data storage /transfer.
- For any 'x'∈V, if NODETYPE(x) = 'task', then SIZE(x) represents the compute cycles required for the execution of the corresponding task.

A dataset node corresponds to a unique dataset in the workflow and represents the data-centric activity for the transfer of the dataset from one host to another. The dataset is treated as an atomic unit for the purpose of storage and data transfer. A task node corresponds to a compute-intensive activity in the workflow. A workflow activity that is both data and compute centric can be modeled by a task node preceded and succeeded by data node. Therefore in

model, the terms dataset (task) and dataset (task) node interchangeably.

We include an edge from the entry task node to each of the input dataset nodes, as if these were all produced as outputs of the entry task. The entry task is a pseudo task with a zero processing time and is introduced to model the start of the workflow. Likewise, an exit task node also represents a pseudo task with zero processing time that models workflow termination.

3.1 MULTIPARTITE NATURE OF THE WORKFLOW

A big data workflow W as described in the preceding section has a multipartite structure revealed by the partitioning Ᵽ as follows:

- Ᵽ ≡ $\{P_i\}$; $1 \leq i \leq m$; where m is the total number of ordered partitions and P_i denotes the i^{th} partition.
- For each pair of partitions P_i and P_j in Ᵽ, $P_i \cap P_j = \phi$, i.e., the partitions are disjoint.
- Each node in V belongs to exactly one partition in Ᵽ.
- For any pair of nodes 'x' and 'y' in any partition P_i; $1 \leq i \leq m$, NODETYPE(x) = NODETYPE(y), i.e., all nodes in a partition are of the same type and there is no edge between nodes in the same partition.

The partitions are indexed in order as follows:

- The first partition P_1 contains the entry node only.
- The last partition P_m contains the exit node only.
- There is no edge in E from a node in P_j to a node in P_i; $1 \leq i < j \leq m$.
- For each node 'x' in any partition P_i; $1 < i \leq m$, there is at least one edge from a node in partition P_{i-1}. This condition implies that each node is placed in the lowest possible numbered partition.

The definition order of partitions in Ᵽ implies the following properties:

Property 1: The number of partitions, |Ᵽ|, is a finite odd number.

Property 2: Even numbered partitions contain only dataset nodes and odd numbered partitions contain only task nodes.

Property 3: There are no edges from a node in an even (odd) numbered partition to nodes in an even (odd) numbered partition.

Property 4: For each node in all even numbered partitions (a partition comprising of dataset nodes), there is exactly one incoming edge and one or more outgoing edges.

Property 5: For each node in all odd numbered partitions, there is at least one incoming edge (except the first partition) and at least one outgoing edge (except the last partition). The first partition has no incoming edge and the last partition has no outgoing edge.

Property 6: For any pair of nodes 'x' and 'y' in V, if $y \in pred(x)$ and '$x' \in P_i$ and '$y' \in P_j$, then $j < i$.

Definition 3: (Usage Scope of a Dataset). The Usage Scope of a dataset node 'x' is defined by an ordered pair of indices (i, j); $1 < i < j \leq m$ where '$x' \in P_i$ and j is the highest index such that there is an edge from 'x' to a node in P_j.

Understanding the usage scope of a dataset allows the release of storage occupied by it at the earliest possible time. If an intermediate dataset produced during a workflow run must be kept in storage for future runs or other workflows, the situation is represented by including an edge from the dataset node to the exit node.

3.2 PARALLEL EXECUTION OF TASKS IN THE WORKFLOW

To minimize the completion time of a workflow, one must execute workflow tasks in parallel as much as possible. Based upon the model of workflow presented so far, it is clear that the tasks belonging to the same partition can be executed in parallel provided that there is sufficient available compute capacity. The following equation gives the maximum possible degree of parallelism in a workflow:

Maximum Possible Parallelism = max $|P_i|$ ($1 < i < m$; i is odd); where $|P_i|$ is the number of nodes in partition P_i.

The actual parallelism may be restricted by available compute capacity or by design to minimize the delays due to dataset transfers. Thus, there is a tradeoff between exploiting parallelism and minimizing dataset transfers. Furthermore, these tradeoffs must follow the resource constraints of the workflow execution environment. Figure 1 shows a workflow graph.

3.3 WORKFLOW EXECUTION PLAN

A workflow execution plan lays out the mapping of hosts to nodes in the workflow graph and the relative timings when a task/data transfer is scheduled to start and complete, and when the resources occupied by a node can be released. The relative timings in the plan depend on the processing and data transfer times which in turn depend on the mapping of hosts to nodes in the workflow graph.

Definition 4: (Big data Workflow Execution Plan). A big data workflow execution plan is formally defined as a tuple $\omega \equiv (\Phi, \Gamma)$, where

- Φ is a mapping function such that for each node 'x' in V, $\Phi(x)$ = 'h' where 'h'$\in$H.
- Γ is a mapping function such that for each node 'x' in V, $\Gamma(x) = \{start_x, finish_x, release_x\}$. All times are relative to the start time of entry node. Γ (entry node) = {0, 0, 0}.

To calculate the makespan of an execution plan in partition-based approach we need to obtain maximum storage (requestS) and compute requirements (requestC) of a host at any point within an execution plan. Function 1 computes requestS and request of a host in any step of an execution plan.

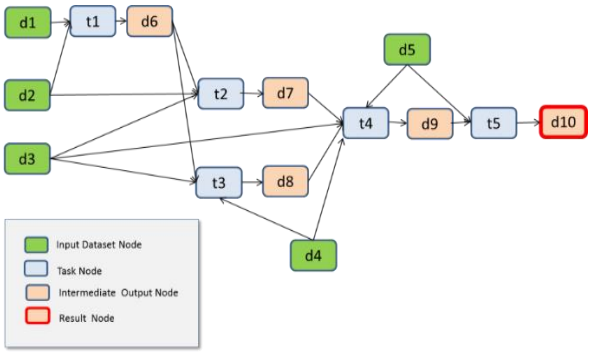

Figure 1: A data-centric workflow with five tasks, five input datasets, four intermediate datasets, and one output set.

Function 1. Computing requestS and requestC for a host

Input:
ω : Execution Plan ω, host '*h*'∈H,
h∈H: host

Output:
maximum storage and compute requirements of host *h*, *requestS*(*h*) and *requestC*(*h*)

```
1.  BEGIN
2.    nodes = storageCounts = computeCounts = {};  // set of counts
3.    counted = {};  // set of node
4.    countS = countC = 0;
5.    For each node 'x' ∈ H
6.      If Φ(x) = 'h' then
7.        nodes ← 'h';      // add 'h' to nodes
8.      End If
9.    End For
10.   For each node 'x' ∈ node
11.     If ('x' ∉ counted) then
12.       counted ← 'x';
13.       countS = SIZE('x');
14.       countC = 1;
15.       For each node 'y' ∈ nodes and 'y' ≠'x'
16.         If (('y' ∉ counted) And (( start_x ≤ start_y ≤ release_x) Or ( start_y ≤ start_x ≤ release_y)))
17.           countS = countS + SIZE('y');
18.           countC = countC + 1;
19.           counted ← 'y';
20.         End If
21.       End For
22.       storageCounts ← countS;
23.       computeCounts ← countC;
24.     End if
25.   End For
26.   requestS = max{storageCounts};      //maximum of all storage counts
27.   requestC = max{computeCounts};      //maximum of all compute counts
28.   return requestS(h), requestC(h);
29. END
```

An execution plan must be **valid** in order to be realized in a workflow. For an execution plan to be valid, it must satisfy the following constraints:

- Hosting Restrictions
- Host Type Constraints
- Storage Capacity Constraints
- Compute Capacity Constraints
- Precedence Constraints

These constraints are formally defined below.

Constraint 1: (Hosting Restrictions) – The execution plan ω satisfies the restricted mapping constraint if and only if for each node 'x' in V, if Φ(x) = 'h' then 'h' ∉ restrictedHosts(x). A mapping policy may restrict a node to be mapped to certain hosts or the host for a given node is predetermined and fixed. For example, certain dataset in raw forms may not be allowed to cross national boundaries during the makespan process.

Constraint 2: (Host Type Constraints) – The execution plan ω satisfies the host type constraint if and only if for each node 'x' in V, Φ(x) = 'h', then either NODETYPE(x) = 'data' and TYPE(h) ≠ 'task' or NODETYPE(x) = 'task' and TYPE(h) ≠ 'data'.

Constraint 3: (Storage Capacity Constraints) – The execution plan ω satisfies the storage capacity constraint if and only if requestS(h) < ASC(h) for each host 'h' in H in the plan ω. Function 1 shows the procedureemployed to compute requestS(h).

Constraint 4: (Compute Capacity Constraints) – The execution plan ω satisfies the compute capacity constraint if and only if requestC(h) < ACC(h) for each host 'h' in H in the plan ω. Function 1 shows the procedure to compute requestC(h).

Constraint 5: (Precedence Constraints) – The execution plan ω satisfies the precedence constraint if and only if for each pair of nodes 'x' and 'y', if there is an edge from 'x' to 'y' in E then $start_y > finish_x +$ transfer(x, y), and $release_x > finish_y$ where transfer(x, y) = (SIZE(x) + SIZE(y)) × DTR(Φ(x), Φ(y)).

3.4. WORKFLOW EXECUTION PLANNING PROBLEM

Definition 5: (Makespan of an Execution Plan) - Given an execution plan ω, the makespan of ω, makespan(ω), is the total execution time of the workflow.

Definition 6: (Makespan Minimization Problem) – Given a workflow graph **W** and the workflow execution environment **Ω**, a makespan minimization problem is formally defined as the problem of finding a workflow execution plan $\omega^{optimal}$ such that i) $\omega^{optimal}$ is a valid execution plan, and ii) for all valid execution plans ω, $makespan(\omega^{optimal}) \leq makespan(\omega)$.

4. SOLUTIONS FOR WORKFLOW EXECUTION PLANNING

In this section two main solutions for finding heuristic solutions to the workflow execution problem formulated in section 3.4 are described. These solutions follow completely different approaches which are subsequently compared and evaluated by experiments. First approach is based on construction of execution plan one at a time in the ascending order. Algorithm 1, also called *WEP-A*, describes the solution based on the first approach. Second approach, called *WEP-B* is described in Algorithm 2 and is based on a meta-heuristic optimization technique.

4.1 ALGORITHM 1: *WEP-A*

The *WEP-A* algorithm works on the input workflow laid out as a finite set of partitions. Workflow task nodes belong to odd numbered partitions and dataset nodes to even numbered partitions. Figure 2 represents partitioned layout of the workflow shown in Figure 1. irst *WEP-A* algorithm computes, for each partition, the set of preferred, permissible and other hosts for each workflow node host (lines 6-12). A valid mapping of workflow nodes to the available hosts is then computed next (lines 13-18). Following that, start and finish times of each workflow nodes in the plan are calculated (lines 19-26). The unused datasets are released in order to increase the storage capacity of the hosts (lines 27-31). Finally, the algorithm calculates the makespan of the execution plan and returns both an execution plan and its makespan as outputs (lines 35-43).

P#	Workflow	Mapping Function
P_1		$\Phi(t_{entry})$ = 'client'
P_2		$\Phi(d_1),\Phi(d_3) = h_1$ $\Phi(d_2) = h_3$ $\Phi(d_4) = h_2$
P_3		$\Phi(t_1) = h_4$
P_4		$\Phi(d_6) = h_3$
P_5		-----
P_6		------
P_7		------
P_8		-----
P_9		-----
P_{10}		$\Phi(d_{10}) = h_5$
P_{11}		$\Phi(t_{exit})$ = 'client'

Figure 2: Workflow of Figure 1 shown as Ordered Partitions.

Algorithm 1. WEP –A : Incremental Generation of Execution Plan by Partitions

Input:

W(P, E, SIZE): Workflow graph where $\mathbb{P} = \{P_i\}$; $1 \le i \le m$, *Ω(H, TYPE, ASC, ACC, DTR, CR):* Workflow Execution environment

Output:

Execution Plan ω, Makespan(ω)

1. **BEGIN**
2. $finish_{entry\ node} = start_{entry\ node} = release_{entry\ node} = 0$;
3. $\Phi(x)$ = 'client' if x is t_{entry} or t_{exit} , otherwise $\Phi(x) = \phi$;
4. $i = 2$; // $P_1 = t_{entry}$ and $P_m = t_{exit}$
5. **For each** P_i $(1<i<m)$ **do**
6. sorted_node_list = nodes in P_i sorted by descending order of SIZE;
7. **While** sorted_node_list $\neq \phi$ **do**
8. x = remove next node in sorted_node_list;
9. **If** $\Phi(x) \neq \phi$ **then break;** // *this will be true for nodes with pre-existing fixed allocation*
10. *permissible_hosts* = {h: $h \in$ H **And** h$\notin$ restrictedHosts(x) **And** (**if** NODETYPE(x) == 'Data' **then** (TYPE(h) $\neq$ 'Task' **And** SIZE(x) $\le$ ASC(h)) **else** TYPE(h) $\neq$ 'Data' **And** SIZE(x) $\le$ ACC(h))};
11. *preferred _hosts* = *permissible_hosts* $\cap${$\Phi(y)$: $y \in$ *pred*(x)};
12. *other_hosts=permissible_hosts - preferred _hosts*;

// *find a valid mapping*

13 **If** *preferred _hosts* $\neq \phi$ **then**

14. **If** (NODETYPE(x) ==’Data’)
15. $\Phi(x)=\underset{h}{\text{argmin}}\, DTR(\Phi(y), h)$ for all $h \in$ *preferred _hosts*
16. **Else**
17. $\Phi(x)=\underset{h}{\text{argmin}}\, CR(h)$ for all $h \in$ *preferred _hosts*
18. **End if**
19. **End if**

// *compute the scheduled start and finish timings in the plan*

20. **If** (NODETYPE(x) == 'Data') **then**
21. $start_x = finish_y$: $y = pred(x)$;
22. $finish_x = start_x + (SIZE(x) \times DTR(\Phi(y),\Phi(x)))$;
23. $ASC(\Phi(x)) = ASC(\Phi(x)) - SIZE(x)$;
24. **Else** //x is a task node
25. $start_x$ = max ($finish_y$ + $SIZE(y) \times DTR(\Phi(y), \Phi(x))$) : y$\in$*pred*($x$);
26. $finish_x = start_x + (SIZE(x) \times CR(\Phi(x))$;
27. $release_x = finish_x$;

// release predecessor dataset resources, if possible

28. **For each** y$\in$ *pred*(x)
29. **If** there exists node 'z' such that y$\in$*pred*(z) **And** $\Phi(x) = \phi$ **then**
30. $release_y = finish_x$;
31. $ASC(\Phi(y)) = ASC(\Phi(y)) - SIZE(y)$;
32. **End If**
33. **End for**
34. **End while**
35. **End While**

// compute makespan – include datasets that need to be transferred back to client

36. **For each** y$\in$ *pred*(*exit node*)
37. **If** $release_y$ == null **then** //must be a data set
38. $release_y = finish_y + (SIZE(y) \times DTR(\Phi(y),$'client'$))$;

//transferring to the client node may not be needed for all nodes.

39. **End if**
40. **End For**
41. $finish_{exitnode;}$= max ($release_y$): y∈ *pred(exit node);*
42. makespan(ω) = $finish_{exitnode}$
43. return ω and makespan(ω)
44. **END**

4.2 Algorithm 2: *WEP-B*

Algorithm 2. *WEP-B*: Execution Planning by Evolutionary Optimization

Input:
W: Workflow Graph;
Ω(H, TYPE, ASC, ACC, DTR, CR): Workflow Execution environment,
population_size: size of population,
elitism_rate: rate of elitism,
mutation_rate: rate of mutation,
num_iterations: number of iterations

Output:
Execution Plan ω and its makespan

1. **BEGIN**

   num_elite ← *population_size* × *elitism_rate;*

   num_crossover ← *(population_size – num_elite)/2;*

   num_mutation ← *population_size* × *mutation_rate;*

   Populatio← { }; $Population_{new}$← { }; $Population_{temp}$← { };

   /* Step 1: Initialize population */

2. **For** *i = 1* **to** *population_size* **do**
3. Generate a new random mapping, Φ, which produces a valid execution plan , ω ≡ (Φ, Γ);
4. $Population \leftarrow Population \cup \{< \omega, Makespan(\omega) >\}$;
5. **End For**
6. **For** *i=1* **to** *num_iterations* **do**

   /* Step2: Carry over *num_elite* best execution plans to next generation */

7. $Population_{new}$ ← add top *num_elite* execution plans from *Population* in the ascending order of makespan;

   /* Step 3: Generate *num_crossover* new execution plans by crossover */

8. **For** *j=1* **to** *num_crossover* **do**
9. Select randomly a pair of execution plans say ω_A and ω_B from *Population*;
10. Generate valid execution plans ω_C and ω_D by an appropriate one-point crossover of mapping functions in ω_A and ω_B ;
11. $Population_{temp}$←$Population_{temp}$ ∪ {< ω_C , *Makespan* (ω_C)>, < ω_D , *Makespan* (ω_D)>};
12. **End for**

   /* Step 3: Mutate *num_mutation* execution plans after crossover*/

13. **For** *j=1* **to** *num_mutation* **do**
14. select a new execution plan, ω_A, randomly from $Population_{temp}$;
15. mutate the mapping function in ω_A to generate a new execution plan, ω_B ;
16. Replace < ω_A , *Makespan* (ω_A)> in $Population_{temp}$ by < ω_B , *Makespan* (ω_B)>
17. **End for**
18. $Populatin \leftarrow Population_{new} \cup Population_{temp}$; $Population_{new} \leftarrow \{\}$; $Population_{temp} \leftarrow \{\}$;
19. **End for**
20. return the execution plan ω in *Population* with the shortest makespan;
21. **END**

Algorithm *WEP-B* proceeds by generating an initial population of valid execution plans. The size of the population *population_size* is an input parameter to the algorithm. Execution plans along with their makespans are retained for the next step. The algorithm is then iterated a number of times as described by another input parameter *num_iterations*. Alternatively, the algorithm can be terminated by checking a convergence criterion (for example, percentage change in average makespan) at the end of each iteration. In each iteration, three tasks are performed. First, a fixed number of execution plans with the shortest makespans are selected to be carried over to the next generation. Then we randomly cross-over pairs of execution plans to generate new valid plans. Finally, we modify plans from the previous step by controlled mutations that result in valid plans only. The new population is then formed by the union of carried over plans and genetically modified plan. In the end the execution plan with the shortest makespan is returned.

5. EXPERIMENTS

The two proposed workflow execution planning strategies have been simulated in a cloud computing environment on Wayne State University's high performance Grid network. The simulation used eight grid nodes and compared the two algorithms by simulating five synthetic workflow applications based on five real data-centric workflows: Montage, CyberShake, Epigenomics, LIGO and SIPHT (Bharathi et al., 2008). These workflow applications are represented in Figure 3 and were developed through the Pegasus workflow management system for different research domains like bioinformatics and astronomy. For these experiments each of the selected workflows was run 200 times with different parameters as shown in Table 2. They were also compared with a random solution generator *WEP-R*. In addition, we evaluated the performance of *WEP-B* performance with 20% of fixed-location datasets.

Figure 4 shows the workflow makespans produced by varying the number of hosts and fixing the number of workflow nodes. The size of the workflow makespans are increased by increasing the number of workflow nodes in all three strategies. As expected, the WEP-A and WEP-B strategies generate execution plans with shorter makespans on average than the WEP-R strategy. Furthermore, the gap increases as the number of hosts increases. This is because a random strategy may spread out the workflow and thereby introduce dataset transfer delays. Algorithm WEP-B seems to perform better than WEP-A. This can be explained by the simpler structure of workflows. As the complexity and size of workflow graph increases, it is expected that WEP-B is likely to perform substantially better than WEP-A. It can also be noticed that, WEP-A and WEP-B's performances flatten out where the excess availability of additional hosts no longer impacts the makespans anymore.

Table 2. Description of dataset and hosts of our experiment.

Overall workflow nodes and hosts	
# of nodes (datasets and tasks)	[50,200,750,3000]
Dataset size	[1TB – 100TB]
Task computation	[10Hz – 10^3Hz]
# of hosts (datasets and tasks)	[5,10,25,50,100]
Data transmission rate	[0.1MBps – 3.0MBps]
Data host storage capacity	[200TB – 1PB]
Task host computation rate	[10^3Hz – 10^6Hz]

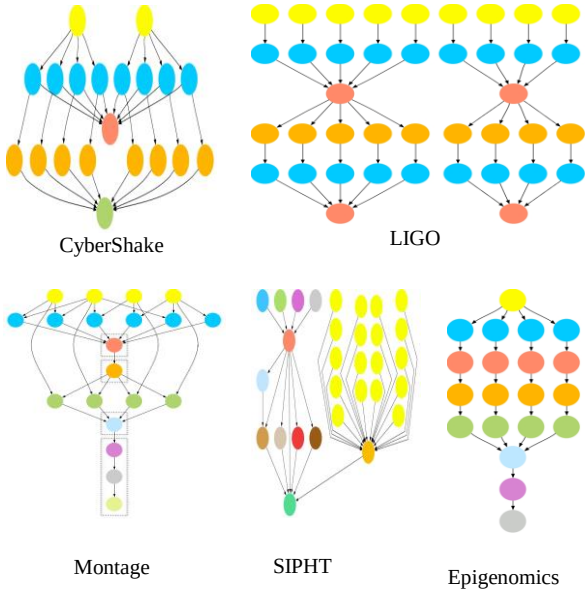

Figure 3: The structure of five realistic data-centric workflows [Bharathi et al., 2008]

Figure 5 shows similar results of experiments in which the number of hosts are fixed whereas the number of nodes in the workflow graph can vary.

In a third set of experiments (shown in Figure 6), the locations of the varying percentages of the datasets are fixed, By fixing the locations of datasets, workflow makespans show an increase for the WEP-A and WEP-B algorithms whereas there is almost no change for the WEP-R strategy. This behavior is attributed to the fact that by fixing the locations of datasets, there is less freedom to reduce the makespans by reallocating datasets.

6. RELATED WORK

Cultural Algorithms (CA) is a branch of evolutionary computation inspired from social evolution. It is composed of a knowledge component called belief space as well as the population component. CA have been successfully applied to various single or multi-objective optimization problems (Reynolds 1999, Jayyousi and Reynolds 2014).

Previous research work in the context of distributed computing environments has been mainly focused on the performance modeling and

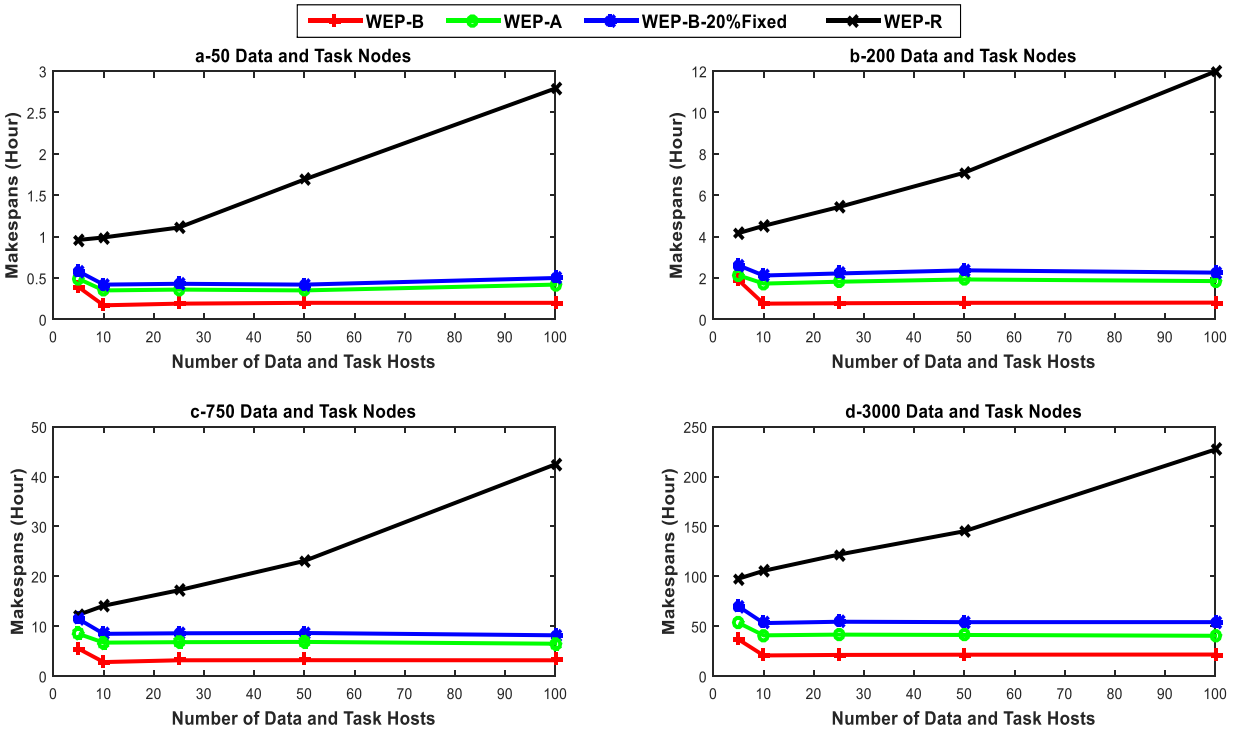

Figure 4. Workflow makespans by varying the number of data/ task hosts.

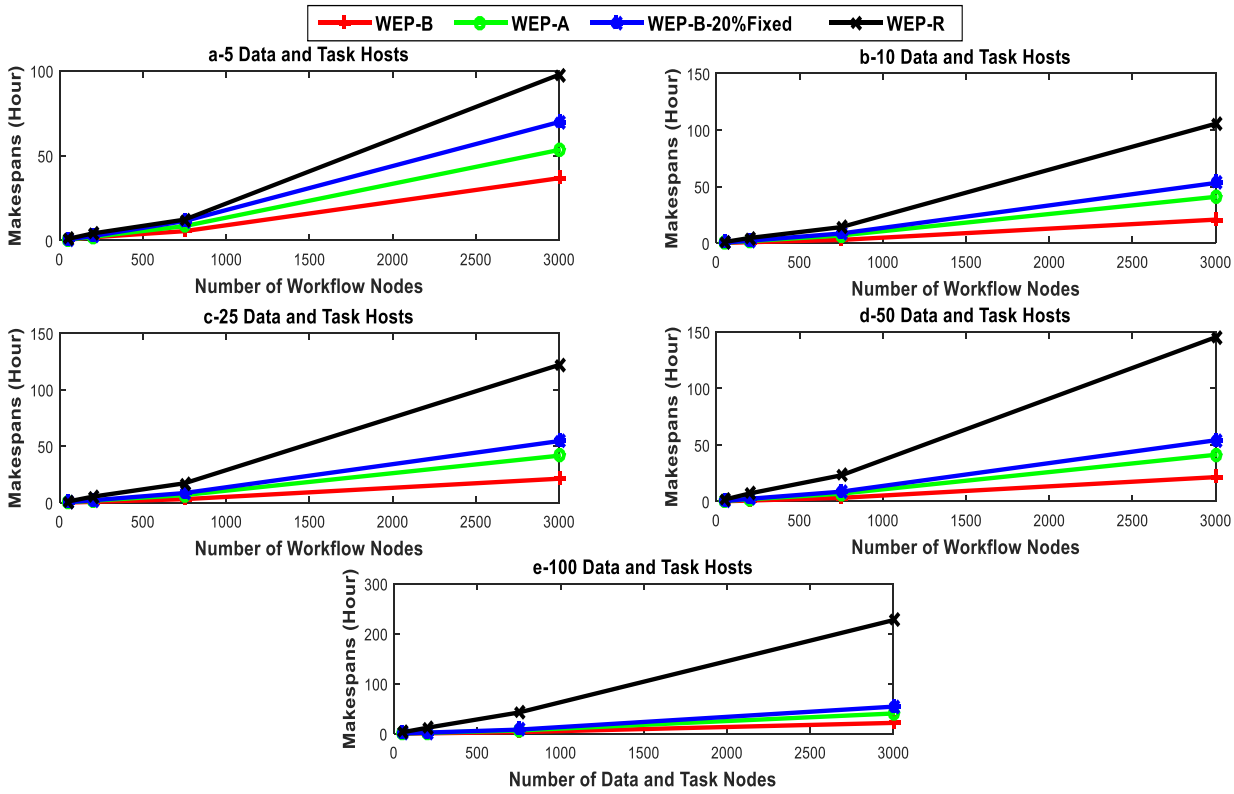

Figure 5. Workflow makespans by varying the number of workflow nodes (data/task).

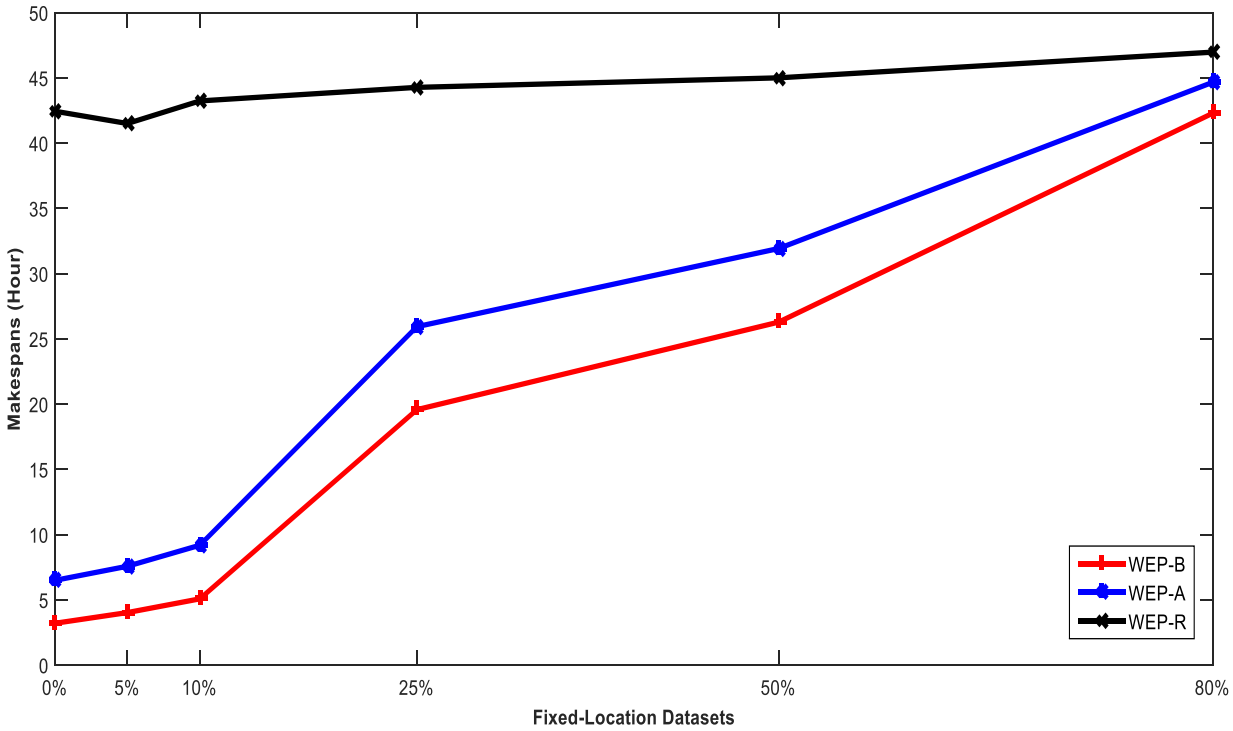

Figure 6. Workflow makespans by varying the percentages of fixed-location datasets for fixed-size of workflows with 1000 nodes and 50 hosts.

A optimization of job scheduling and task allocation. Due to the emergence of e-science and big data workflows, it has become important to consider the costs of dataset allocation and movement while developing an execution plan for a big data workflow. This is even more crucial when a big data workflow is executed in a distributed environment that involves multiple and heterogeneous data centers. Kosar et al., (2005a, 2005b) proposed an allocation framework for distributed computing systems which considered the data placement subsystem as an independent module along with the computation subsystem. In their proposed model, both data placement and task computation jobs can be queued, scheduled, monitored, managed and even check pointed. Kayyoor et al., (2013) modelled the data placement and replication strategies for the distributed environments. They stated that minimizing query latencies is not a critical issue in many analytical workload scenarios. So they tried to minimize the average number of computational nodes deployed for the workflow by grouping the most interdependent data together based on their occurrences of their common query accesses. Chervenak et al., (2007) explored the advantages of separating data placement services from workflow management systems. By applying an autonomous data placement service along with a data replication service, they demonstrated the benefits of pre-staging data when compared to the data stage in and out strategies of the Pegasus workflow management system. Lin and Lu (2011) proposed an algorithm for mapping a data-centric workflow application into a cloud execution environment where resources can be dynamically acquired using the elastic services of the cloud.

In Çatalyürek et al., (2011) workflows were modeled as hypergraphs and a hypergraph partitioning technique, k-way partitioning, was proposed to minimize the cut size. In that way, they were able to cluster the workflow tasks along with their required data in the same execution site. Yuan et al., (2010) applied a heuristic binary clustering algorithm to pre-cluster datasets and greedily assigned workflow tasks to an execution site which contained the most input datasets for that workflow. Although their approach placed the most interdependent data sets together and can reduce data movement, it did not work as well with clusters of different sizes and different densities. The other related work is Er-Dun et al., (2012) where they

applied Genetic Algorithm to heuristically produce their data allocation solution along with incorporating load balancing as part of optimization criterion. Their model reduced data movement but workflow task allocation was not considered.

This paper extends the BDAP work of Ebrahimi et al., (2015) through the incorporation of the following new to the model:

1. The workflow makespan minimization problem was formulated in terms of a big data environment and workflow. This was not addressed in the BDAP paper.
2. Two new allocation strategies, WEP-A and WEP-B, were added to the extended BDAP model. While BDAP focused on minimizing data movement during workflow execution, WEP-A and WEP-B focused on the minimization of the workflow makespans.
3. A series of extensive experiments were employed to study the performance of WEP-A and WEP-B in comparison to WEP-R (a random solution) using five real-world workflows.

7. CONCLUSIONS

In a workflow that involves big datasets, either as inputs or intermediate outputs, its makespan can vary greatly depending on how the tasks and datasets are allocated in a distributed computing environment. This paper provided a formal definition of the makespan minimization problem for big data workflows, and proposed two efficient workflow execution planning strategies. In particular, two algorithms *WEP-A* and *WEP-B* were proposed. Each followed a different allocation strategy. *WEP-A* followed a phased approach to generate an execution plan whereas *WEP-B* used an evolutionary optimization strategy to find a valid plan with the shortest makespan. Both of these strategies were evaluated and compared through extensive simulation experiments by varying workflow graphs and resources in the workflow environment. Our experiments demonstrated that *WEP-B* performed better than *WEP-A* although *WEP-B* was simpler and faster than *WEP-B.* In more complex and larger scaled workflows, the improvements due to evolutionary optimization in *WEP-B* were likely to be become even more pronounced.

ACKNOWLEDGMENT

The authors thank Dr. Satyendra Rana for technical assistance and ideas in preparing the paper. This work is supported by National Science Foundation, under grants NSF ACI-1443069. This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812.

REFERENCES

Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H. and Vahi, K. (2008). "Characterization of scientific workflows." In Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS), pp. 1-10.

Çatalyürek, Ü. V., Kaya, K., and Uçar, B. (2011). "Integrated Data Placement and Task Assignment for Scientific Workflows in Clouds." In Proceedings of the 4th international workshop on Data-intensive distributed computing, pp. 45-54.

Chebotko, A., Lin, C., Fei, X., Lai, Z., Lu, S., Hua, J., Fotouhi, F. (2007). "VIEW: A Visual Scientific Workflow Management System." In Proceedings of the IEEE international Workshop Scientific Workflows, pp. 207-208.

Chervenak, A., Deelman, E., Livny, M., Su, M. H., Schuler, R., Bharathi, S., Mehta, G. and Vahi, K. (2007). "Data Placement for Scientific Applications in Distributed Environments." In Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, pp. 267-274.

Deelman, E. and Chervenak, A. (2008). "Data Management Challenges of Data-Intensive Scientific Workflows." In Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID), pp. 687-692.

Deelman, E., Singh, G., Su, M. H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C. and Katz, D. S. (2005). "Pegasus: A Framework for Mapping

Complex Scientific Workflows onto Distributed Systems." Scientific Programming, vol. 13, iss. 3, pp. 219-237.

Ebrahimi, M., Mohan, A., Kashlev, A., and Lu, S. (2015). "BDAP: A Big Data Placement Strategy for Cloud-Based Scientific Workflows." In Proceedings of the 1st IEEE International Conference on Big Data Computing Service and Applications (BigDataService), pp. 105-114.

Er-Dun, Z., Xing-Xing, X. and Yi, C. (2012). "A Data Placement Strategy Based on Genetic Algorithm for Scientific Workflows." In Proceedings of the 8th International Conference on Computational Intelligence and Security (CIS), pp. 146-149.

Freire, J., Silva, C. T., Callahan, S. P., Santos, E., Scheidegger, C. E. and Vo, H. T. (2006). "Managing Rapidly-Evolving Scientific Workflows." In Provenance and Annotation of Data. Springer Berlin Heidelberg, pp. 10-18.

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P. and Oinn, T. (2006). "Taverna: A Tool for Building and Running Workflows of Services." Nucleic Acids Research, vol. 34, iss. Web Server issue, pp. 729-732.

Jayyousi, T., Reynolds, R.G. (2014) "Extracting Urban Occupational Plans Using Cultural Algorithms." In Proceedings of the IEEE Computational Intelligence Magazine, Vol. 9, No. 3, pp. 66-87.

Juve, G. and Deelman, E. (2010). "Scientific workflows and clouds." Crossroads, vol. 16, no. 3, pp. 14-18.

Kashlev, A. and Lu, S. (2014). "A System Architecture for Running Big Data Workflows in the Cloud." In Proceedings of the IEEE International Conference on Services Computing (SCC), pp.51-58.

Kayyoor, A. K., K., Deshpande, A. and Khuller, S. (2013) "Data Placement and Replica Selection for Improving Co-location in Distributed Environments." CoRR abs/1302.4168.

Kosar, T., Son, S., Kola, G. and Livny, M. (2005a). "Data Placement in Widely Distributed Environments." Advances in Parallel Computing 14, pp. 105-128.

Kosar, T. and Livny, M. (2005b). "A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems." Journal of Parallel and Distributed Computing (JPDC), vol. 65 no. 10, pp.1146-1157.

Lim, C., Lu, S., Chebotko, A., Fotouhi, F. (2010). "Prospective and Retrospective Provenance Collection in Scientific Workflow Environments." In Proceedings of the IEEE International Services Computing (SCC), pp. 449-456

Lin, C. and Lu, S. (2011). "SCPOR: An Elastic Workflow Scheduling Algorithm for Services Computing." In Proceedings of the International Conference on Service-Oriented Computing and Applications (SOCA), pp. 1-8.

Lu, S., and Zhang, J. (2009). "Collaborative Scientific Workflows." Web Services, ICWS. IEEE International Conference on, pp. 527-534

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E., Tao, J. and Zhao, Y. (2005). "Scientific Workflow Management and the Kepler System." Concurrency and Computation. Practice and Experience, Special Issue: Workflow in Grid Systems, vol. 18, iss. 10, pp. 1039–1065.

Reynolds, R.G. (1999) "An Overview of Cultural Algorithms." New Ideas in Optimization, D. Corne, F. Glover, and M. Dorigo Ed., McGraw Hill Press, pp. 367-378.

Yuan, D., Yang, Y., Liu, X. and Chen, J. (2010). "A Data Placement Strategy in Scientific Cloud Workflows". Future Generation Computing Systems, vol. 26, no. 8, pp. 1200-1214.

Zhao, Y., Hategan, M., Clifford, B., Foster, I., Von Laszewski, G., Raicu, I., Praun, T. S. and Wilde, M. (2007). "Swift: Fast, Reliable, Loosely Coupled Parallel Computation." In Proceedings of the IEEE International Workshop on Scientific Workflows, pp. 199-206.

Authors

Mahdi Ebrahimi is a PhD candidate in the Department of Computer Science, Wayne State University. His research interests include Big Data, Big Data Workflows, Big Data Placement, Big Data Workflow Scheduling and NoSQL Databases. He has published several research articles in peer-reviewed international conferences, including IEEE conference on Big Data and Big Data Service. He is a member of IEEE and ACM.

Aravind Mohan is a PhD candidate in the Department of Computer Science, Wayne State University. His research interests include Big Data Management, NoSQL Databases, Cloud Computing and Services Computing. He has published several research articles in peer-reviewed international conferences, including IEEE conference on services computing, big data, big data computing services and applications and the ACM SIGIR conference. He is a member of IEEE and ACM.

Andrey Kashlev is a PhD candidate in the Department of Computer Science, Wayne State University. His research interests include Big Data, NoSQL Databases, Cloud Computing, and Services Computing. He has published several papers in peer-reviewed international journals and conferences, including Data and Knowledge Engineering, International Journal of Computers and Their Applications and the International Conference on Services Computing. He is a member of IEEE.

Shiyong Lu is an associate professor in the Department of Computer Science, Wayne State University, and the director of the Big Data Research Laboratory. Dr. Lu received his Ph.D. in computer science from Stony Brook University in 2002. Dr. Lu's current research interests focus on big data workflows, big data modeling, and provenance management. Dr. Lu is an author of two books and over 120 articles published in various international journals and conferences, including IEEE Transactions on Services Computing (TSC), Data and Knowledge Engineering (DKE), IEEE Transactions on Knowledge and Data Engineering (TKDE). He is the founding chair of the IEEE International Workshop on Scientific Workflows (SWF) and a Co-Editor-in-Chief of the International Journal of Cloud Computing and Services Science. He is a founding editorial board member of International Journal of Big Data and a senior member of the IEEE.

Dr. Robert G. Reynolds received his Ph.D. degree in Computer Science, specializing in Artificial Intelligence from the University of Michigan, Ann Arbor. He is currently a professor of Computer Science and director of the Artificial Intelligence Laboratory at Wayne State University. He is also an Adjunct Associate Research Scientist with the Museum of Anthropology at the University of Michigan-Ann Arbor. His interests are in the development of computational models of cultural evolution for use in the simulation of complex organizations and in computer gaming applications. Dr. Reynolds produced a framework, Cultural Algorithms, which is a data intensive evolutionary search algorithm based upon principles of social and cultural evolution. He has applied this approach to solving data intensive problems and has received funding from both government and industry to support his work. He has published over 250 papers on the evolution of social intelligence in journals, book chapters, and conference proceedings. He is currently an associate editor for the IEEE Transactions on Artificial Intelligence in Games, and IEEE Transactions on Evolutionary Computation. Dr. Reynolds is a senior member of the IEEE and a member of the ACM.

hipore.com

BIG DATA

A Technical Publication of the Services Society

ISSN : 2326-4411 (Print)
ISSN : 2326-442X (Online)