Adjacent Matrix based Deduction for Grid Workflow Applications*

Fan Zhang[1], Junwei Cao[2,3], Lianchen Liu[1,3], Cheng Wu[1,3]
[1]*National CIMS Engineering and Research Center, Department of Automation*
Tsinghua University, Beijing 100084, P. R. China
[2]*Research Institute of Information Technology, Tsinghua University, Beijing 100084, P. R. China*
[3]*Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, P. R. China*
Corresponding Email: jcao@tsinghua.edu.cn

Abstract

This paper presents an adjacent matrix based method for verifying and analyzing large scale workflows, which is used to find our three basic prosperities: sequential, parallel and simple mixing. Any workflow with the three characteristics can be grouped and deducted into smaller and even easier workflow. This reduction can also be used to figure out branches in complicated connected workflows. We exemplify our method using a real scientific workflow in our previous work. The analysis of reduced workflow can further facilitate the verification and validating process.

Key words: Adjacent matrix, graph deduction, scientific workflow

1. Introduction

Cloud computing[1,2] is a buzzword in internet with its far-reaching vision to revolutionize the internet applications into large scale data centers with virtualization and visualization. In this way, the execution of applications, either operation system based or user based, are fully deployed and regulated in remote servers with experts and experienced works who are dedicated to ensure the service could satisfy different user's demand. Workflow applications, which has long been proven to successfully organize and direct various applications, can be still useful for dynamic and complicated services.

Scientific workflow[3,4,8] is emerging as a method to ensure large scale and complicated grid applications running smoothly, flexibly and reliably [5]. The functionalities of traditional grid workflow enabling technologies, such as job scheduling, dispatching, etc are thus enhanced. However, as the even more events and interoperations among multiple events, the organization of the resources, customers, and domain users are even more difficult.

The performance of validating and verifying large scale workflows is still a difficult and error-prone work [7]. Since as the increase of the workflow scale, the uncertainties, unpredictabilities, and even more unregular connections make this work very difficult to follow. This problem becomes even worse in evolving and dynamic workflow with constrained to memory usage and network provision.

To solve the above difficulties, there are many attempts to visualize the workflow structure and try to find some characteristics in common. Some researches focus on finding simple yet easy-to-figure-out ways [9], such as single connections between two large sub-workflows, to decompose a large one into several small ones. Along this line, many other researches [6] has proposed methods to cut and/or add branches between pairs of activities in order to find common structures.

This paper proposes a novel and interesting algorithm to simplify the representation of a very complicated workflow. It uses Adjacent matrix to describe the relationship among activities and finds out the basic structures, such as sequential, parallel and simple mixing, of such workflows. Then we derive a small Adjacent matrix with more and better view of a complicated workflow. Besides that, our proposed can be used very easily to figure the branches in a workflow application, which thus can be used as an enhanced method to supplement the above related researches.

The rest of the paper is organized as follows. In Section 2, We introduce some basic concepts and propositions. Section 3 and 4 introduce that we decompose a large workflow Adjacent matrix into s small one and pick out its branches. The case study using a scientific workflow is implemented in section 5 to exemplify our work. Section 6 concludes the paper.

2. Preliminaries

2.1 Basic Concepts

All the abbreviations and symbols in the following definitions and based on figure 1.

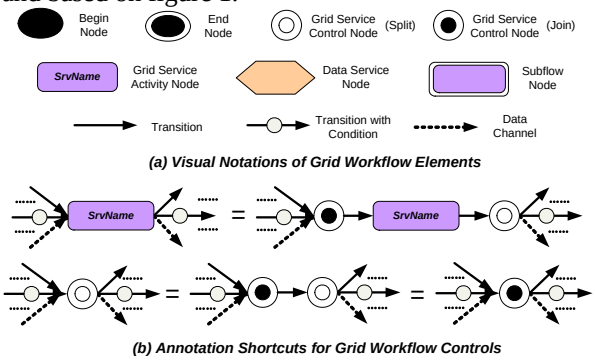

Figure 1. Visualization of grid workflow elements

Definition 1 (Region): Two Srv&Ctrl nodes or Begin / End nodes N_{head} and N_{tail} form a region in a grid workflow Γ, denoted by {N_{head}, N_{tail}}, IFF: (1) ∄ N_{head} →N_1→…N_m→ End where End is the End node of Γ, and N_i != N_{tail} (i=1,…,m); (2) ∄ Begin →N_1→…N_m→ N_{tail} where Begin is the Begin

node of Γ, and N_i != N_{head} (i=1,…,m). N_{head} / N_{tail} *are called* Start Region / End Region *respectively*.

Definition 2 (Maximized Region): Two *Srv&Ctrl* nodes or *Begin / End* nodes N_{head} and N_{tail} form a *maximized region* in a grid workflow Γ, IFF ∀ *Begin* →N_1→…N_m→ *End* where *Begin*, *End* are the *Begin* node and the *End* node of Γ, N_{head} and N_{tail} are contained in the path and N_{head}≠N_{tail}. N_{head} / N_{tail} *are called Maximized* Start Region / End Region *respectively*.

Definition 3 (Decomposition): A maximized region $\{N_1, N_2\}$ in Γ is said to be *decomposable* IFF: (1) there are more than one path N_1→…→N'→…→N_2 s.t. N_1 can reach N_2 in Γ and ∃N'⊂$\{N_1, N_2\}$, s.t. $\{N_1, N'\}$ or $\{N', N_2\}$ is a maximized region; or (2) there is only one path N_1→…→N'→…→N_2 s.t. N_1 can reach N_2 in Γ and for any N_0 → N_1 and N_2 → N_3, either $\{N_0, N_2\}$, $\{N_1, N_3\}$ or $\{N_0, N_3\}$ is a maximized region. Moreover, for nodes N'_1, … N'_m within region $\{N_1, N_2\}$, the set of maximized regions {{N', N''} | {N', N''}={N_1, N'_1} or {N'_1, N'_2} or … or{N'_m, N_2}} is said to be a *total decomposition* of $\{N_1, N_2\}$ IFF all {N', N''}s are maximized regions and are not further decomposable.

Definition 4 (Standard Region): A maximized region $\{N_{start}, N_{end}\}$ in Γ is a *standard region* IFF $\{N_{start}, N_{end}\}$ belongs to the total decomposition of Γ. N_{start} / N_{end} *are called* Standard Start Region / End Region *respectively*.

2.2 Basic Propositions

Proposition 1 (Sequential Combination): Two Srv&Ctrl nodes N1 and N2 form a Sequential Region in a grid workflow Γ, denoted by N_{start} – N_1 – ⋯ –N_2 – N_{end}, IFF: (1) N_{start} / N_{end} are Standard Start / End Region ; (2) ∀ N_i *in* $\{N_1, N_2\}$. OutDegree(N_i) = InDegree(N_i) = 1.

This proposition makes sure all nodes within a standard region (exclude the Standard Start / End Region) can be sequentially ordered then generalized and replaced by a single node. Figure 2 shows the detailed procedure of the simplification.

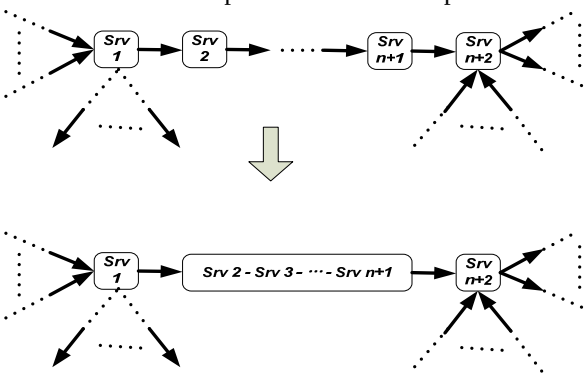

Figure 2. Sequential Combination

Prososition 2 (Parallel Combination): Two *Srv&Ctrl* nodes N_1 and N_2 form a Parallel Region in a grid workflow Γ, denoted by N_{start} – N_1 / ⋯ /N_2 – N_{end}, IFF: (1) N_{start} / N_{end} are Standard Start / End Region ;(2) ∀ N_i *in* N_1 – ⋯ – N_i – ⋯ – N_2, there should one and only one node N_i ; (3) ∀ N_i *in* N_1 – N_i – N_2. OutDegree(N_i) = InDegree(N_i) = 1.

This proposition makes sure all nodes within a standard region (exclude the Standard Start / End Region) can be parallel then generalized and replaced by a single node. Figure 3 shows the detailed procedure. Notice should be made that in above constraint (2) in definition 2, N_i can be original Srv&Ctrl node or combination of nodes based on all these rules.

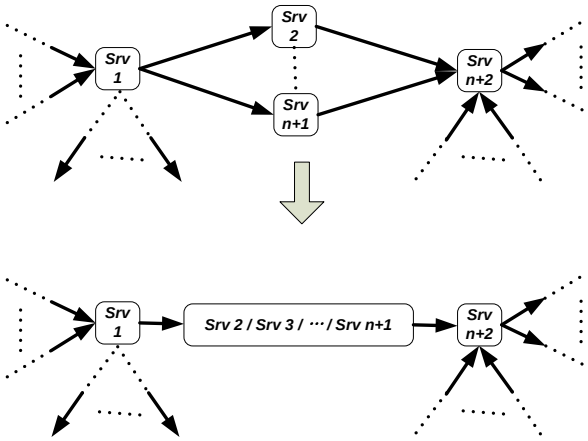

Figure 3. Parallel Combination

Proposition 3 (Simple Mixing Combination): Two *Srv&Ctrl* nodes N_1 and N_2 form a Simple Mixing Region in a grid workflow Γ, denoted by N_{start} – (*N_1* / ⋯ /N_2) – N_{end}, IFF: (1) N_{start} / N_{end} are Standard Start / End Region ;(2) ∀ N_i *in* N_1 – ⋯ – N_i – ⋯ – N_2, there should one and only one node N_i ; (3) ∀ N_i *in* N_1 – N_i – N_2. OutDegree(N_i) = InDegree(N_i) = 1. (4) O*ne single connection exists between* N_{start} *and* N_{end} *directly.*

This proposition make sure all nodes within a standard region (exclude the Standard Start / End Region) can be simply mixed and then replaced by a single node. Figure 4 shows the detailed procedure of the simplification. Notice that in above constraint (2) in definition 2, N_i can be original Srv&Ctrl node or combination of nodes based on all these rules.

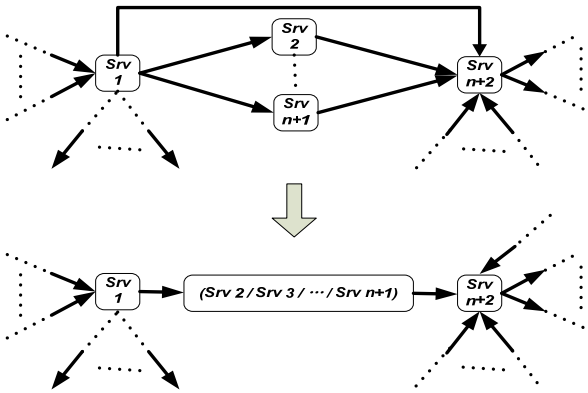

Figure 3. Simple Mixing Combination

3. Workflow Indexing, Laying and Deduction

3.1 Workflow indexing and laying

Topological Sort in directed acyclic graph is widely used nowadays to order all vertices to analyze the structure and components. Two typical commonly used methods are BFS (Breadth-First-Search) and DFS (Depth-First-Search). Neither of these methods is suitable for analyzing grid workflows since their adjacent matrices shows only one aspect of the intrinsic characteristic of the structure. BFS deals more with the horizontal information of one same layer while DFS concerns mainly on the hierarchical relationships among the nodes. In our proposed indexing and laying methods, we tend to combine the two analytical component together to better understand the inner information. Together with that, all these reorganizing and ordering are implemented within the scale of one region. Both advantages in the two traversing algorithms should be fully utilized and a better combination can give us more direct information of the graph.

In order to standardize the structure of one graph, index of layers should be adopted firstly and properly calculated. Here is the algorithm we propose to index and lay one graph.

Algorithm 1: graph indexing and laying

Input: $\mathcal{G} = <\mathcal{N}, \mathcal{A}>$, start

$//\mathcal{N} = \{n_1, n_2, \cdots, n_{|\mathcal{N}|}\}\ \mathcal{A} = \{(n_i, n_j)\}\ |i, j = 1, \ldots, |\mathcal{N}|$

Output: Index of each node $\{1,2,\cdots, |\mathcal{N}|\}$

Procedure:

```
Queue Q ← null
List N ← 𝒩
Int index ← 0
InQueue(start)
   while (NotNull (Q))
      n_p = DeQueue (Q)
      n_p.index = index +1
      𝒩 = 𝒩 - n_p
       for all (node n_t in 𝒩) do     // n_t parent node n_k son node
         if InEdge(n_t, n_k) && NotherPath(n_t, n_k)
          then InQueue(n_k)
       end for
       for all (node n_k in Q) // n_d branch parent node
          for all (node n_d in 𝒩)
             if InEdge(n_d, n_k)
             then  n_s.index = index +1
                    𝒩 = 𝒩 - n_s
          end for
       end for
   end while
end
```

The algorithm shows the steps for re-indexing and laying for the original graph which is useful for us to better understand the horizontal and vertical relationships in the grid workflows. It paves the way for us to know the regions, maximized regions and standard regions easily and directly. Together with that, we could figure out the branch from its adjacent matrix introduced bellow and give further analysis of the ruled structure.

3.2 Information from Adjacent Matrix

After re-indexing and laying for all the nodes included in our grid workflows, the detailed structure and hierarchy can be easily derived. We introduce adjacent matrix, AM = (mij) |N| ×|N| with respect to G , which is a |N|×|N| boolean matrix to show the relationship of all nodes. It is defined like this: iff (ni, nj) < A, mij = 1 or else mij = 0.

From our proposed indexing and laying method, the adjacent matrix is more orderly arranged and reveals more information of the inner structure.

Definition 4(Sub-diagonal matrix): A square matrix $(\mathcal{SDM})_{k\times k}$ is called one sub-diagonal matrix of square matrix $(\mathcal{SM})_{n\times n}$ IFF: (1) $\exists\, t \in \{1,2,\cdots,n-k\}$, $\forall$ i, j $\in \{1,2,\cdots,k\}$, $(\mathcal{SDM})_{(i,j)} = (\mathcal{SM})_{(t+i,\, t+j)}$. We denote the relationship as $(\mathcal{SDM})_{k\times k} \in_{sdm} (\mathcal{SM})_{n\times n}$.

Proposition 4 (Standard Region in adjacent matrix): Given the complete adjacent matrix $(\mathcal{AM})_{|\mathcal{N}|\times|\mathcal{N}|}$, $(\mathcal{SDM})_{k\times k}$ is called the standard region of $\mathcal{AM}$ IFF: (1) $\mathcal{SDM} \in_{sdm} \mathcal{AM}$, (2)if $(\mathcal{SDM})_{(i,j)} = (\mathcal{SM})_{(t+i,\, t+j)}$, t $\in \{1,2,\cdots, |\mathcal{N}|-k\}$ and i, j $\in \{1,2,\ldots,k\}$, matrix $\{(t+k, t+k+1),(t+k-1, |\mathcal{N}|)\}$ should be all zeros. Matrix{(a,b) ,(c,d)} denotes the rectangular matrix start from diagonal element (a,b) and end to diagonal element (c,d).

Proposition 5 (Sequential Region in adjacent matrix): Flow $N_{start} - N_1 - \cdots - N_m - N_{end}$ form one sequential region iff the corresponding adjacent matrix $(\mathcal{SRM})_{(i,j)}$ satisfy if j = i+1 $(\mathcal{SRM})_{(i,j)} = 1$ else $(\mathcal{SRM})_{(i,j)} = 0$. i,j$\in \{0,1,\ldots, m-1\}$.The related predefined indices are N_{start}.index = 0, N_t.index = t (t = 1,…, m)and N_{end}.index = m+1. These indices apply to Proposition 6 and 7.

Proposition 6 (Parallel Region in adjacent matrix): Flow $N_{start} - N_1 / \cdots / N_m - N_{end}$ form one parallel region iff the corresponding adjacent matrix $(\mathcal{PRM})_{(i,j)}$ satisfy if i=0 and j$\in$ {1,…, m } or j = m+1 and i$\in$ {1,…, m} $(\mathcal{PRM})_{(i,j)} = 1$ else $(\mathcal{PRM})_{(i,j)} = 0$.

Proposition 7 (Simple Mixing Region in adjacent matrix): Flow $N_{start} - (N_1 / \cdots / N_m) - N_{end}$ form one simple mixing region iff the corresponding adjacent matrix $(\mathcal{SMRM})_{(i,j)}$ satisfy if i=0 and j$\in$ {0,1,…,m+1} or j = m+1 and i$\in$ {0,1,…, m+1} $(\mathcal{PRM})_{(i,j)} = 1$ else $(\mathcal{SMRM})_{(i,j)} = 0$.

Proposition 8 (Inter-Region Connected in adjacent matrix):

Suppose Sub-diagonal matrix of R $(\mathcal{SDM})_{(i,j)} = (\mathcal{SM})_{(t+i,t+j)}$ and n is the dimension $\mathcal{SDM}$, region R in grid workflow is called to be inter-connected iff matrix{(t+n+1,t+1),(|$\mathcal{N}$|,t+n-1)} is zero matrix.

Proposition 9 (Intra-Region Connected in adjacent matrix): Suppose Sub-diagonal matrix of R_1 $(\mathcal{SDM})_{(i,j)} = (\mathcal{SM})_{(t+i,t+j)}$, R_2 $(\mathcal{SDM})_{(i,j)} = (\mathcal{SM})_{(k+i,k+j)}$ and n_1 is the dimension $\mathcal{SDM}$ of R_1 and n_2 is the dimension $\mathcal{SDM}$ of R_2. Two Regions R_1 and R_2 in grid workflow is called to be intra-connected iff matrix{(t+n+1,t+1),(|$\mathcal{N}$|,t+n-1)} is zero matrix.

3.3 Workflow Deduction

In order to simplify complex grid workflows, especially those composed of large scale and frequently time variant structure, say, scientific workflows and so on, deduction is the primary concern we should take. It is impossible to make a unified deduction rule that apply to any structure. Instead, we fully utilized the three simple structure introduced in previous chapters to cluster similar and closed related workflow structure and merge them together to form a much smaller and more easily analyzable flow pattern.

The primary work we have to do is renaming the new grouped element and record the history of deduction. Some symbols and denotations previously used are our choice to form a simple element in complex workflows. The steps should be made recursively to form the most simple and undeductable workflow structure, it is called Base Workflow Structure (BWS). Mathematically, such BWS is isomorphic to numerous other structures that share the similar flow pattern. Similar analytical method can be adopted to implement those isomorphic workflows.

Our algorithm firstly deduct all sequential regions and such deduction is recorded and replaced by one single sequential node, then all parallel regions are fully investigated to be deducted to similar single parallel node, notice should be made that parallel deduction utilize the nodes not only from original ones , but also those sequential or parallel nodes. Simple mixing structure should be then deducted further. All these three steps are not necessary to be strictly ordered since their functionalities are almost the same. Several rounds of such deduction should be made to reach to the BWS, in which the general structure is unchangeable after another round of three types of deduction. The procedure is introduced in Algorithm 2.

Algorithm 2: Workflow Deduction

Input: $\mathcal{G} = <\mathcal{N}, \mathcal{A}>$, start

$//\mathcal{N} = \{n_1, n_2, \cdots, n_{|\mathcal{N}|}\}\ \mathcal{A} = \{(n_i, n_j)\}\ |i, j = 1, \ldots, |\mathcal{N}|$

Output: $\mathcal{G} = <\mathcal{N}', \mathcal{A}'>$, start

$//\mathcal{N}' = \{n_1, n_2, \cdots, n_{|\mathcal{N}'|}\}\ \mathcal{A} = \{(n_i, n_j)\}\ |i, j = 1, \ldots, |\mathcal{N}'|$

Procedure:

```
Name ← null
CurFlow ← G
OldFlow ← G
while (!Equal(OldFlow,CurFlow))
    OldFlow = CurFlow
    CurFlow = SeqDeduction(CurFlow)
    CurFlow = ParDeduction(CurFlow)
    CurFlow = SimDeduction(CurFlow)
end while
```

Sub Algorithm 2: Sequential / Parallel / Simple Mixing Deduction

Input: Initial Workflow Iniwf

Output: Deducted Workflow Dedwf

$N_{1.name}$ ← null

Procedure:

```
Switch(DeductionType)
  Case SeqDeduction:
    if(IsSeqStart(N_start) && IsSeqEnd(N_end))
      for all (node N_i in{ N_start, N_end })
        N_i.name += " - " + N_i.name
        DeleteEdge((N_i,*) and (*,N_i))
  Case ParDeduction:
      if(IsParStart(N_start) && IsParEnd(N_end))
        for all (node N_i in{ N_start, N_end })
          N_i.name += " / " + N_i.name
          DeleteEdge((N_i,*) and (*,N_i))
  Case SimDeduction:
      if(IsSimStart(N_start) && IsSimEnd(N_end))
        N_i.name += "("
        DeleteEdge((N_start, N_end))
        for all (node N_i in{ N_start, N_end })
          N_i.name += " - " + N_i.name
          DeleteEdge((N_i,*) and (*,N_i))
          N_i.name += ")"
 end
```

4. Total Decomposition Based on Adjacent Matrix

In [10], one algorithm, *totaldecomposition* gives out the way to decompose complex workflows by combining Aspect Oriented Programming(AOP) into region analysis, which successfully totally decompose workflows into its maximized regions$\{M_1, M_2, \cdots, M_n\}$. However, it is really complex to analyze the procedure and not very easily spot out regions, not to say standard regions or other structure information. The adjacent matrix, on the other hand, gives out very intuitive information of how to totally decompose the workflow such that it is even easier for us to do further verifications. Proposition 8 gives the characteristics of regions and parallel branches, which is our principles to do the decomposition. Here is the algorithm of adjacent matrix based total decomposition.

Algorithm 3: Total decomposition based on adjacent matrix

Input: $(\mathcal{AM})_{|\mathcal{N}| \times |\mathcal{N}|}$

Output: $\{M_1, M_2, \cdots, M_n\}$

Procedure:

```
ip ← 0
region_num ← 0
```

count_num ← 0
  while(count_num < $|\mathcal{N}|$)
    region_num++
    M_{region_num} = GetMatrixMember($\mathcal{AM}$, count_num, count_num + SDM(ip))
    count_num = count_num + SDM($\mathcal{AM}$, ip)
  end while
end procedure

Sub Algorithm 3: Sub-diagonal Matrix(SDM)
Input: adjacent matrix ($\mathcal{AM}$) $_{|\mathcal{N}|\times|\mathcal{N}|}$
Initial position ip
Output: Matrix dimension n
Procedure:
Step ← 0
index = FindMaxColumnIndex($\mathcal{AM}$[ip,index] equalto 1)
Step ← index
while (Matrix {(index+1,ip),($|\mathcal{N}|$, index-1)} is not zero matrix)
  forall i in 0 to index
index = FindMaxColumnIndex(Matrix {(index+i,ip),($|\mathcal{N}|$, index-1)})
  Step ← index
end while
return Step
end procedure

In the algorithm above, Function SDM () is used to derive the dimension of the current standard region. They are those tasks in the pipelines which can be grouped together. That is the critical sub-function in order to tell whether the current task is the end of the current standard region. FindMaxColumnIndex (Matrix) is used to calculate the value which is not zero and in the same time with the maximized column. Since based on our proposition 8, no edge of standard region should connect to outer regions. That is to say one region should be completely self-contained in a cubic matrix. Based on the return dimension, we can easily recover the included members inside the region from function GetMatrixMember($\mathcal{AM}$, count_num, count_num + SDM(ip)), which is used to get those tasks from $\mathcal{AM}$[count_num] to $\mathcal{AM}$[count_num + SDM(ip)].

5. Case Study

5.1 Scientific Workflow — LIGO

In order to verify the requirements that describes the properties of certain demands, one example of scientific workflows called gravitational waves detection is introduced here to show the efficiency of the proposed methods.

Gravitational Waves (GW) are produced by the movement of energy in mass of dense material which fluctuate space-time structure. The analysis of unknown mass movement and formulation in the universe is stemmed from its detection. But the difficulty is that the detection and analysis of them relates to multiple tasks and massive data.

LIGO (Laser Interferometer Gravitational-Wave Observatory) includes three most sensitive GW detectors in the world which are L1, H1, H2 introduced in the following examples jointly built by Caltech and MIT. LIGO Scientific Collaboration (LSC) includes over 500 research scientists from over 50 institutes all over the world who are working hard on LIGO data analysis for GW detection. LIGO produces one terabyte of data per day and LIGO data analysis require large amount of CPU cycles. The LIGO data grid[8] provides such a computing infrastructure to integrate petabytes of data storage capability and thousands of CPUs and enable research collaboration cross multiple institutes.

A typical example of a grid workflow for LIGO data analysis can be found in [10]. Figure 4 describes one workflow structure of GW search and its visualization.

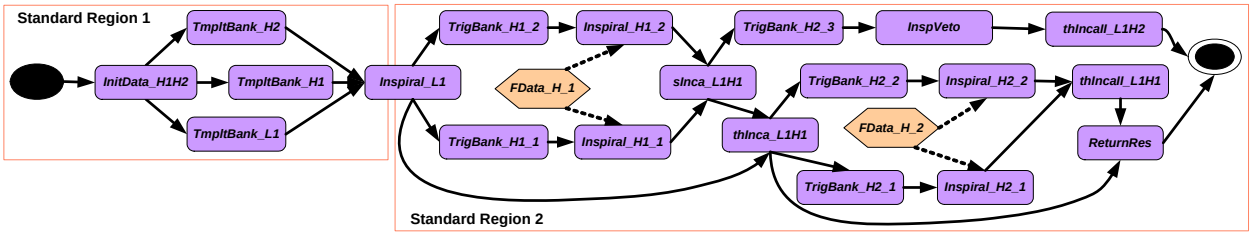

Figure 4. Scientific workflow for LIGO data analysis

Firstly three detectors sample their signal data (*FData*) respectively, we use orange hexagon to show this step since it is the preparation step rather than the formal step. Then Initializing data (*InitData*) is implemented to divide the sampled data into smaller blocks for further analysis. We can use some theoretical models to produce a series of (*TmpltBank*) that marches with the detected data, then the undesirable data can be easily excluded though the matching of *TmpltBank* and the blocks, this step is called *Inspiral*, Those data satisfy the requirement should be combined with other detectors to go through coincidental analysis (*sInca* and *thInca*) for noise discovery. Those signals that pass the coincidental analysis should be reput into the *TmpltBank* for optimizing the performance in later steps (*TrigBank*). Since all these work should be done when the interferometers are in stable status, it is necessary to veto those bad signals (*InspVeto*) based on the status of all interferometers when they are operating. In the practical gravitational waves analysis, the veto of bad signals are frequently happened do analyze the impulse wave disturbance from extra information channel of interferometer L1, in the end the final signal should again go though coincidental analysis (thIncall) to get the candidate signals of gravitational waves. By means of the procedure, LIGO users are able to derive the gravitational waves that exist potentially around the celestial objects and explain some related phenomenon, say, the movement of them.

In [8],Those blocks that perform specific tasks are grouped together to describe the detection pipeline. Those tasks are

implemented as individual programs and the pipeline itself is then implemented as a logical graph, called a *directed acyclic graph* or DAG, describing the workflow (the order that the programs must be called to perform the pipeline activities) which can then be submitted to a batch processing system on a computer cluster. Condor high throughput computing system [11] can then be used to manage the DAG and to control the execution of the programs. Though it is more or less satisfying those mid-scale pipelines perform, more work should be done to optimize the pipelines' structure and to unburden the load of the analytical instruments when it comes to large-scale workflow structures.

5.2 LIGO workflow deduction and analysis

Based on our proposed indexing and laying methods introduced in section II, we input the LIGO workflow in figure 4, our algorithm automatically indexes all the tasks in the pipelines. Here is the corresponding indices result.

Table 1. Indexes and layers of scientific workflow

Start	0
InitData_H1H2	1
TmpltBank_H2	2
TmpltBank_H1	3
TmpltBank_L1	4
Inspiral_L1	5
TrigBank_H1_2	6
TrigBank_H1_1	7
FData_H_1	8
Inspiral_H1_2	9
Inspiral_H1_1	10
sInca_L1H1	11
TrigBank_H2_3	12
thInca_L1H1	13
InspVeto	14
TrigBank_H2_2	15
TrigBank_H2_1	16
FData_H_2	17
thIncall_L1H2	18
Inspiral_H2_2	19
Inspiral_H2_1	20
thIncall_L1H1	21
ReturnRes	22
End	23

Together with that, our ordered graph can be even clearly laid to form much understandable structure based on the indexing and laying. It is shown in figure 6.

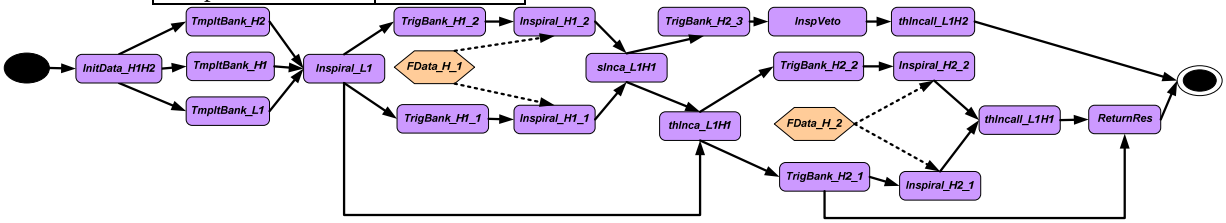

Figure 5. Scientific workflow after indexing and Laying

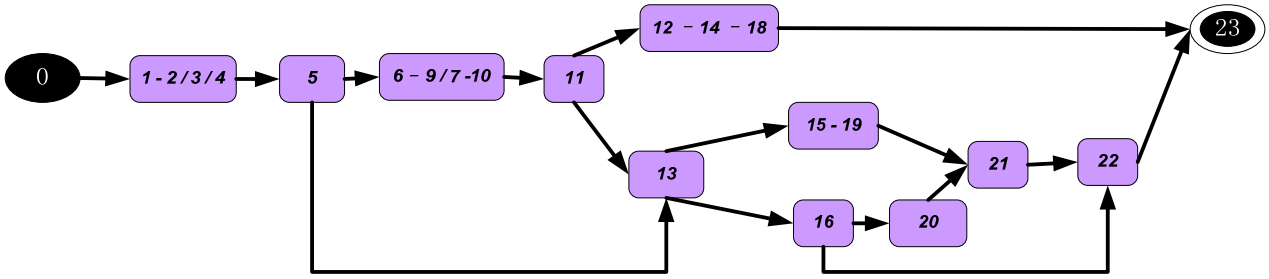

Figure 6. Scientific workflow after deduction

The understandability of the workflow structure is much more readably and the laying of which is quite obvious. Its adjacent matrix is in table 2.

It is quite clear that there are two regions ({0, 5};{5,23}) in the workflow. The second region {5, 23}, if task (5, 13) in the yellow node can be eliminated, can be further decomposed into two other regions. The same rule applies to task (16, 22). From the above matrix, we can also very easily figure out those three types of basic regions: sequential region, parallel region and simple mixing region.

Suppose we decompose the second large region {5,23} into two smaller regions by eliminating the branch (5,13) that connect the two regions. Notice should be made that region {11, 23} is still really complex and more work should be done to decompose it further. Then we should rename the indices in region {11, 23} and together with that indexing and laying a second round of it. More basic regions can be generalized and even simpler workflow structure can be derived.

After the implementation of our proposed algorithms to deduct the workflow into a simplified pattern, the final BWS can be shown in figure 6.

Based on the deducted workflow, 11 tasks still remain in the final BWS, which could not go any further in our proposed deduction rules. We can easily spot out the general structure of the complex workflow and detail out more information from this figure. From this deducted flow, we could do local verification based on the aggregated tasks and extend such verification to even larger regions and finally to the whole

flow. Such verification logics are more understandable and easily operated. Mostly importantly, it helps us to spot out those parallel branches that negatively affect the analysis of the whole workflow. The correspondent adjacent matrix is as follows.

Table 2. Indexes, layers, blocks and branches of our scientific workflow

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0		1																						
1			1	1	1																			
2						1																		
3						1																		
4						1																		
5							1	1						1										
6										1														
7											1													
8										1	1													
9												1												
10												1												
11													1	1										
12															1									
13																1	1							
14																			1					
15																				1				
16																					1		1	
17																				1	1			
18																								1
19																						1		
20																						1		
21																							1	
22																								1
23																								

Table 3. Indexes, layers, blocks and branches of our reduced scientific workflow

	0'	1'	2'	3'	4'	5'	6'	7'	8'	9'	10'	11'	12'
0'		1											
1'			1										
2'				1			1						
3'					1								
4'						1	1						
5'													1
6'								1	1				
7'											1		
8'										1		1	
9'											1		
10'											1		
11'												1	
12'													

6. Conclusions and Future Work

We firstly conclude our works in this paper and then suggest two possible directions to extend this work.

6.1 Conclusions of this work

The more structural the flow structure is, the better performance result is easily derived. Figure 7 gives the implementation procedure of the proposed method in this paper.

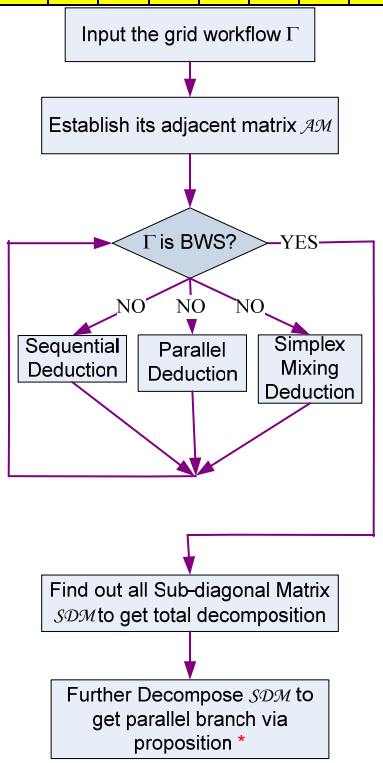

Figure 7. Implementation of graph deduction

6.2 Our Future Work

For further research, we suggest to extend the work in the following two directions:

(1) **Quantify the verifying time reduction**. Since our proposed algorithms can be used to decompose a complicated workflow, the followed work should be justifying the usefulness by measuring verification time deduction. We'll further our work as proposed in [10] to better illustrate the significance of this method.
(2) **Develop toolkits for workflow deduction.** We are designing some toolkits with matlab to build a platform, which is used to automatically convert an input workflow, whether a BPMN or Pi calculus representation, into its adjacent matrix and then followed deduction could be carried out. This work can be better facilitate our analysis in the future.

Acknowledgement

This work is supported by National Science Foundation of China (grant No. 60803017) and Ministry of Science and Technology of China under the national 863 high-tech R&D program (grants No. 2008AA01Z118 and No. 2008BAH32B03).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica and M. Zaharia, Above the Clouds: A Berkeley View of Cloud Computing, Technical Report No. UCB/EECS-2009-28, Feb. 10, 2009.

[2] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The Eucalyptus Open-Source Cloud-Computing System, in Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, China.

[3] J. Cao, S. A. Jarvis, S. Saini and G. R. Nudd, "GridFlow: Workflow Management for Grid Computing", in *Proc. 3rd IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, Tokyo, Japan, pp. 198-205, 2003.

[4] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing", *J. Grid Computing*, Vol. 3, No. 3-4, pp. 171-200, 2005.

[5] K. Xu, Y. Wang and C. Wu, "Ensuring Secure and Robust Grid Applications - From a Formal Method Point of View", *Advances in Grid and Pervasive Computing*, LNCS Vol. 3947, Springer Verlag, pp. 537-546, 2006.

[6] K. Xu, Y. Wang and C. Wu, "Aspect Oriented Region Analysis for Efficient Equipment Grid Application Reasoning", in *Proc. 5th IEEE Int. Conf. on Grid and Cooperative Computing*, Changsha, China, pp. 28-31, 2006.

[7] E. M. Clarke, O. Grumberg and D. A. Peled, *Model Checking*, MIT Press, 1999.

[8] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams and S. Koranda, "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists", in *Proc. 11th IEEE Int. Symp. on High Performance Distributed Computing*, Edinburgh, Scotland, pp. 225-234, 2002.

[9] S. Wang and M. P. Armstrong, "A Quadtree Approach to Domain Decomposition for Spatial Interpolation in Grid Computing Environments", *Parallel Computing*, Vol. 29, No. 10, pp. 1481-1504, 2003.

[10] . K. Xu, J. Cao, L. Liu, and C. Wu. Proc. Performance Optimization of Temporal Reasoning for Grid Workflows Using Relaxed Region Analysis, *in Proceedings of 22nd IEEE Int. Conf. on Advanced Information Networking and Applications Workshops*, GinoWan, Okinawa, Japan, 187-194, 2008.

[11] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in *Practice: The Condor Experience" Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.