Eng Int Syst (2021) 4: 241–252


**Engineering**
***Intelligent Systems***

# Federal Synergy Computing Model Based on Network Interconnection

**Hechuang Wang***

*School of Information Engineering, North China University of Water Resources and Electric Power, Zhengzhou 450045, China*

To solve the shortage problem of the computing power provided by a single machine or small cluster system in scientific research, we offer a collaborative computing system for users. This system has massive operation ability. It introduces a scalable mixed collaborative computing model. Through the internet and heterogeneous computing equipment, the system uses the task decomposition model. This system can solve the research and development problem of the shortage of local computing power. To test the model, a subtask decomposition example is used. The results of the example analysis show that the computing work can obtain the shortest computation time when the number of calculation nodes is more than the number of subtasks; maximum calculation efficiency can be achieved when the number of the calculating nodes approaches the number of subtasks. Through joint collaborative computing, the extensible mixed collaborative computing mode can effectively solve the mass computing problem for the system with heterogeneous hardware and software. This paper provides the reference for the system, which provides large scale computing power through the Internet and addresses the research problems due to the lack of computing ability.

Keywords: Federal Computing, Task Decomposition, Task Scheduling, Data Communication

## 1. INTRODUCTION

Under the impetus of science and engineering applications, many breakthroughs have been made in the computational model with the support of the algorithms and architecture by integrating in modern computing network technology [1–4].

High performance computing capacity through the model provides a powerful foundation platform for the computing of related areas. The simulations of various natural phenomena have reached unprecedented accuracy by using these high performance computing platforms [5], and the platform provides a supercomputing ability to design new drugs to combat emerging viruses and other diseases [6]. Large systems (such as cosmology) and small systems (such as cell research) are hungry for computing power, this is the driving force behind researching computing models, improving system computing power and building large-scale computing platforms.

Computing hardware stacks and parallel system development can provide general-purpose large scale parallel computing capabilities; however, expensive computing hardware and complex system design are difficult to afford in a small scientific computing system. The objective of this research is to design a cost-effective platform for a small scientific computing system in order to provide super computing power. This will solve the issue of insufficient computing power and the platform will be easy to expand and develop with parallel computing systems [7].

This research is devoted to the development of a scalable hybrid federal computing model for building cost-effective, scalable high energy computing systems, in order to provide a computing method for the scientific field, such as the classified system or the special scenario that cannot be applied in other ways such as cloud computing. The research focuses primarily on the construction of a system model that is the core content and the key technology of the system design.

*Corresponding author Email: WangHC123SL@163.com

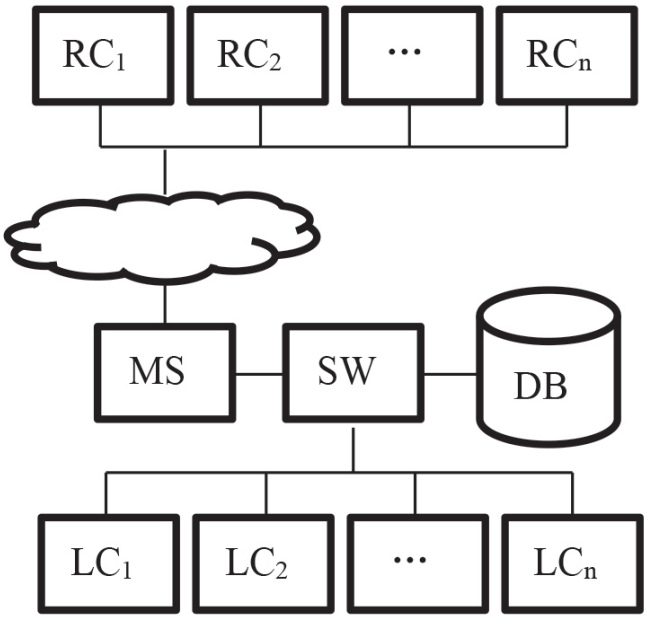


**Figure 1** The System Network Topology.

## 2. DESIGN OF THE SYSTEM ARCHITECTURE

### 2.1 Topology of the system network

Although network computing provides unprecedented computing power for scientific computing, it is difficult to complete large-scale computing due to the lack of scientific research computing equipment and the special requirements in certain fields that network computing cannot provide, such as cloud computing. In these cases, a heterogeneous platform must be built through simple redeploying and connecting existing computing resources with a certain computing scale, or a large amount of funds must be spent to build a private computing platform with large computing power. Motivated by this demand, this research designed a heterogeneous scalable hybrid federal computing model based on network, The overall network topology of the system is shown in Figure 1.

In Fig. 1. the network topology diagram of the system, $LC_{1\sim n}$ are the local computing nodes, the number of which depends on the size of the network IP address allocation pool. Therefore, different types of network will provide a different number of node accesses, and as a result this will constrain the computing capacity of the computing system constructed by them.

DBS provides a data storage system for the system; the whole system can share this data source, with the help of this node the system can complete the data sharing and publishing. MS system is designated as the unique management server, all the computing nodes and the access nodes must be registered on this server. This server is responsible for the establishment of tasks, task assignments and task scheduling, as it is the central core server that is connected both to the intranet and extranet, it builds a connection channel for RC and LC. Through $RC_{1\sim n}$, a remote access client, the MS can be established and asked for computing tasks, and one can also download the COM component from the MS server to join the system, and to accept the MS scheduling. MS, DBS and $LC_{1\sim n}$ are connected together through a switch SW, which is responsible for assigning network addresses to them in turn. In this way, LC and RC do not need the same physical structure or software system, this can shield the difference of the system structure and provide the capability of heterogeneous collaborative computing through the upper layer of software design [8].

### 2.2 Task Processing Flow

The remote device *RC* sends a request to the MS to initiate a task, and the task is submitted in the form of a task description. The design of the task description is shown in Reference [9], and it introduces the concept of multitasking job processing [10], the MS task design flow is shown in Figure 2:

Remote users access the MS via the Internet and submit an application to MS. When the RC application logs in successfully, MS will register the RC identity on the server. The RC will become a remote computing node of MS when it applies for a task from MS. After the application is approved by the server, MS will then issue a general computing task for it. If the RC submits a request to the MS application for computing tasks, MS will review the calculation of the RC application, if the application is accepted, establishment of RC computing tasks will be successful, if it is not accepted, establishment of the task will fail. If the RC does not have a task description, then the job description will be written before loading the job description, the MS task will then be scheduled according to the task description. If the access node of the MS is 0 or the idle node is 0, the system does not have any available resources to perform the task, so the task will be queued, otherwise the task will be assigned to perform. After the end of the computing, the RCs report to the system that the task has been completed, and release the system resources.

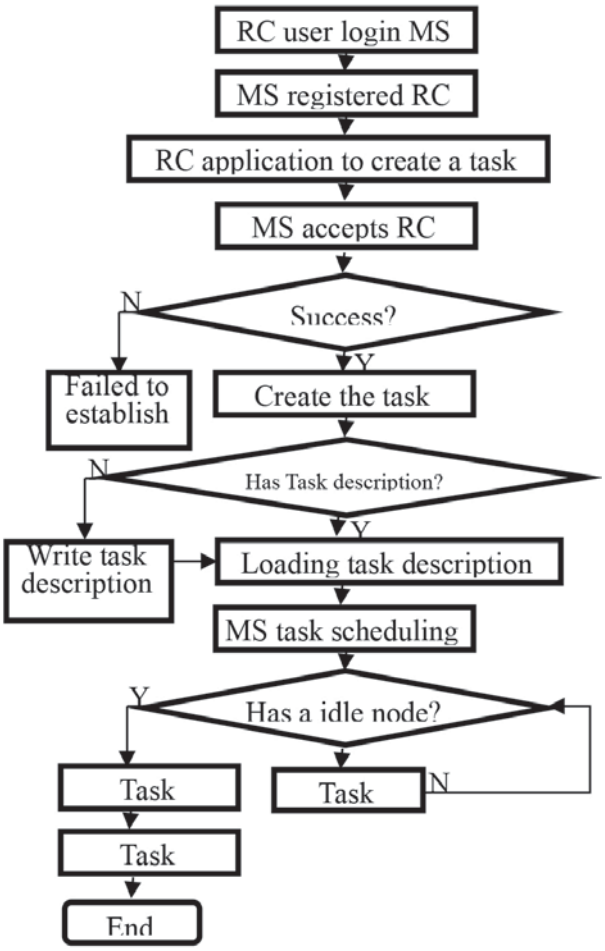


**Figure 2** Task Processing Flow Chart.

According to the needs of the application, the system is constructed by hybrid system architecture. The MapReduce computing model is introduced in the process of assigning tasks to each computing node [11]. The system task specification describes a subset of tasks that split a large problem into a number of small problems, and then perform the tasks on each node in the cluster computing node, otherwise known as a Map process. At the end of the Map process, each node in the cluster will compile, execute and solve the tasks according to the task specification. After the completion of the task, there will be a reduce process, this process will bring all the computing output results of the decomposition of the subtasks together, and send it to the MS and DBS nodes. Whether it is a Reduce process that brings the results together after the system is completed, or the Map process that is executed when the system is initialized, Subtask execution nodes need to communicate with the distribution server for the necessary task descriptions, the task specification which describes the data sources, the remote storage of intermediate key/value results and to submit the results of the implementation. Therefore, it is necessary to provide a large data query and analysis model for the data nodes, and provide remote data access APIs to capture the data of the system design. In order to avoid the accumulation and loss of data, it is necessary to introduce a method to store the new data of the system for when the computing nodes need to save the new generated data in a certain time window. This will ensure different nodes computing performance that is bound to different servers. In order to solve the problem of large data query and analysis, there is a need to calculate the cluster configuration of a small memory computing cluster. There is also a need to introduce a memory computing model to improve the computing performance of a variety of computing models to deal with large data, this model can achieve high real-time data query and analysis rates.

## 3. SUBTASK DECOMPOSITION MODEL AND TASK DESCRIPTION

The system model using the subtask decomposition method is designed according to the reference [12].

Given the computing task $T$, when the complexity of the task $O(T)$ is greater than the given threshold value, continue to resolve the decomposition Subtask $T_i$ of task $T$, $T_i$ can be described by using the task tree view description language (*TTVDL*) based on XML. A list of tasks is created on the basis of task representation, computing task requests from computing node $N$, and establishing the thread of computing nodes. The leaf node $(i)$ is opened after the root traversal calculation based on the tree depth first algorithm.

The task decomposition scheduling algorithm divides the simulation task into 2 layer $m$ fork trees, assigned to each

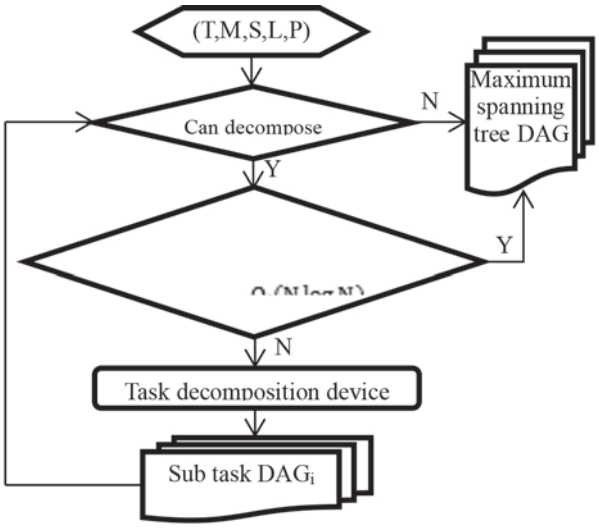


**Figure 3** Task Decomposition Model.

computing unit. If the subtask is larger, it can continue to decompose. The task can be decomposed statically or dynamically. It is necessary to determine the granularity of decomposition, the coefficient of convergence and the convergent boundary of decomposition.

## 3.1 Task decomposition algorithm

A computing task can be described by a task system $(T, M, S, L, P)$. The task decomposition model is shown in Figure 3:

The system uses two layers of nested *DAG*, the *sub_DAG* is a collection of subtasks $DAG_i$ decomposed by *DAG*, $E$ is a collection of communication edge $e_i$, $T$ is a collection of communication costs $T_i$.

Thus we obtained:

$$\begin{cases} DAG = \{sub_{DAG}, E, T, V\} \\ sub\_DAG = \{DAG_1, DAG_2, DAG_3, \ldots, DAG_n\} \\ E = \{e_1, e_2, e_3 \ldots e_n\} \\ T = \{t_1, t_2, t_3 \ldots t_n\} \\ DAG_i = \{subV_i subE_i subT_i subC_i\} \end{cases} . \quad (1)$$

$DAG_i$ is the No. $i$ task in the collection of $t$ subtask, $E$ is a collection of communication edges between $DAG_i$, $C_{ij} \in C$, $C_{ij}$ is a collection of communication costs between $DAG_i$. Because the subtask uses the decomposition method of 2 layer $m$ fork tree, the communication cost will not change because of the task decomposition; the cost is related to the relationship between subtasks and task size; subtask $DAG_i$ is a decision directed acyclic graph, and $DAG_i$ is the second layer of the $M$ fork tree. If $DAG_i$ cannot be decomposed, then $m = 0$, otherwise the value of $M$ is related to the decomposition strategy. $subV_i$, $subE_i$, $subT_i$, $subC_i$ are the collection of $DAG_i$ neutron tasks, the collection of communication edges between subtasks, the collection of sub task completion times, and the collection of communication between the leaf node and the root node. If the number of tasks is $m+1$, the number of communication sides is $M$.

In the whole *DAG* collection, there is a data dependency between $DAG_i$. Data dependency between $DAG_i$ constitutes a dependency collection. Dependency collection is defined as an implementation results collection of subtask $DAG_i$ that requires subtasks $DAG_m, \ldots, DAG_n$ or the transfer variables during the execution of this task. $\{DAG_m, \ldots, DAG_n\}$ is then called the data dependency collection of $DAG_i$. The data dependency collection can be extracted from the task control process according to the subtask function, the subtask $DAG_i$ must be executed after its data dependency collection is executed. There are control dependencies, mutually exclusive relationships, concurrency relationships and interest relationships among subtasks in the whole task decomposition process. When the output of task $DAG_m$ is the input of subtasks $DAG_n$, there is a control dependent relationship between $DAG_m$ and $DAG_n$; When the task $DAG_m$ is runing, the $DAG_n$ subtasks cannot be performed, and when the task $DAG_n$ is runing, the $DAG_m$ subtasks cannot be performed, It shows that there is a mutually exclusive relationship between subtask $DAG_m$ and subtask $DAG_n$; When the $DAG_m$ and $DAG_n$ subtask can be executed at the same time, and the execution of a subtask does not affect the execution of another subtask, then the subtask $DAG_m$ and $DAG_n$ constitute a concurrent relationship; when the implementation of the subtask $DAG_m$ can improve the efficiency and quality of the subtask $DAG_n$, then the subtask $DAG_m$ and subtask $DAG_n$ constitute an interest relationship.

In order to satisfy the data dependency between $DAG_i$, the first root traversal must be performed. In order to solve this problem, first the collection of previous node and the collection of the next node must be solved. Let $PreviousNode(i)$ be the collection of previous node, and $NextNode(i)$ be the collection of next node, therefore:

$$\begin{cases} PreviousNode(i) = \{j | e_{ji} \in E\} \\ NextNode(i) = \{j | e_{ij} \in E\} \end{cases} \quad (2)$$

As a 2 layer $m$ fork tree, task $DAG_i$ has explicit previous and subsequent relationships between each task, therefore, it does not need to seek the relationship between the subtasks.

### 3.2 Define subtask convergence boundary

In order to reduce the transmission of the original data, reduce the traffic and improve the network throughput, a copy of the original data is saved in the access unit $m_i$, the *MI* can be either a computer or a computing independent network unit composed of several computers. The first layer of the task can be extracted from the original data copy of the local computing unit; it does not require data transmission. The original data and the final results are stored in the $S_i$ of data center *DBS*.

Set $M$ as a collection of computing unit $m_i$ in the system, $S$ is a collection of data center $s_i$, $L$ is a collection of computing unit capacity $L_i$, $P$ is a collection of computing power $p_i$.

Thus we obtained:

$$\begin{cases} M = \{m_1, m_2, m_3, \ldots, m_m\} \\ S = \{s_1, s_2, s_3, \ldots, s_s\} \\ L = \{l_1, l_2, l_3, \ldots, l_m\} \\ P = \{p_1, p_2, p_3, \ldots, p_m\} \end{cases} \tag{3}$$

$m_i$ is the No. $i$ unit in the collection of computing units, $s_i$ is the No. $i$ unit in the collection of storage units, $l_i$ corresponds to the load capacity of the computing unit $m_i$, $p_i$ corresponds to the computing power of the computing unit $m_i$

When the system task is decomposed into an $m$ fork tree with hierarchical structure, the tree has a total of $N$ subtasks, the complexity O is introduced, which reduces the complexity from $O(N^2)$ to $O(N \log N)$ [13]. Therefore:

$$kl_i \leq \frac{O_i(N \log N)}{p_i}$$

In the formula, $K$ is the coefficient of convergence. Given the $k$ value, when the ratio of the decomposition subtask complexity is matched with computing power less than or equal to the given boundary convergence condition $kl_i$, then decomposition is stopped, and the decomposition tree is sent to the computing unit $m_i$.

### 3.3 Task decomposition description

The system uses task descriptions to describe the task decomposition, task allocation, task recovery and so on, with each task corresponding to a task description. The nodes involved in the computation need to get the task description from the server and compile it locally. When the computing node LC is ready, the ready signal is sent to the management server, waiting for system scheduling. The management server MS maintains a task description for each computing task, generating a task computing dictionary. The MS implementation processor schedules the tasks by polling the task descriptions, calculating the task dictionary and querying the status of each computing node. Reference [9] used XML as a task description method; this research also uses the XML task tree view to describe the task when designing the task description of the system. The task specification base node is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<TaskDescription>
  <TaskDividedTree> </TaskDividedTree>
  <SubTaskMapping>
   <ComputingNode treeID="">
        <ImportData></ImportData>
        <ExportData></ExportData>
        <NodeDependence></NodeDependence>
    <ComputingCode></ComputingCode>
   </ComputingNode>
  </SubTaskMapping>
</TaskDescription>
```

The <TaskDividedTree/> Node is the static description of the whole task decomposition tree. Each Node contained in the node has a strict description of the communication edge $e_i$, the communication cost $t_i$ and others of the subtask $DAG_i$. The node's $Node(i)$ hierarchical relationship reflects the relationship between the previous and next node, this node is the basis of task scheduling. Node <SubTaskMapping/> is the input, output, static description and calculation method of dependence of each sub node, the number of sub nodes that are described in <TaskDividedTree/>, the <SubTaskMapping/> will contain a description of the number of tasks that do not exceed <TaskDividedTree/>, TreeID is the computing node <ComputingNode/> association Key between <TaskDividedTree/> and <SubTaskMapping/>. Node <ImportData/> contains the input requirements of the computational tasks, and Node <ExportData/> contains the final results. The results of the calculation of the node will be uploaded to the storage server after the computing completion. The node will be recovered before rescheduling when the task computing is completed, and the results of the last task before the recovery will still be maintained in the node. This model can provide P2P node data access, which can reduce server data transfer pressure. <NodeDependence/> is a collection of dependencies of nodes. By accessing the nodes, the nodes can be set to wait, sleep, and wake up and so on. <ComputingCode/> is the algorithm description of the computing nodes. According to this algorithm, the computing nodes are dynamically compiled and calculated locally. The algorithm is compiled only when the first load is run, the second run does not require compilation, this differs from the interpretation of the implementation, so the performance loss can be ignored.

## 4. TASK SCHEDULING ALGORITHM AND MODEL EVALUATION

### 4.1 Task scheduling algorithm

The system algorithm based on the original task scheduling algorithm in Reference [10] has been significantly improved. The improved model uses a hybrid strategy, and its algorithm is described as follows.

In order to improve the efficiency and throughput of the cluster, the task allocation must be intelligent when scheduling a group of tasks, so that the computing resources of each computing node can be fully utilized. In order to prevent some computing tasks from being permanently executed, the equalization of the computing resources as much as possible in

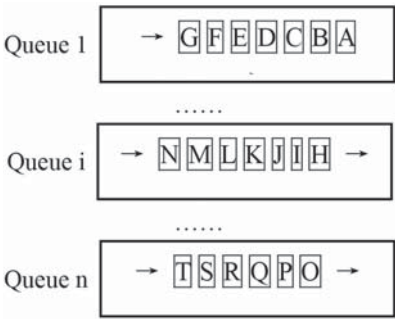


**Figure 4** Algorithm Principle.

each task group must be considered during the system design process.

A task will go through seven states from task submission to completion, such as wait, Map, ready, execute, reduce and complete. When a computational task is successfully created, it needs to be submitted to the system, the system first checks the completeness of the task description, and follows the task instructions with an itemized audit verification. Each LC is queried according to the task description of the sub task description tree. When the idle LC are not able to satisify the computing task it is necessary to wait for the non-idle computing nodes to complete their tasks; the submitted task enters the wait state at this time. According to the task description tree, each sub task will be mapped to each local computing node LC when the system has the idle LC to meet the computing task, then the submitted task then enters the ready state. The implementation state begins when the management server then assigns each node in turn to start the calculation according to the instructions of the dependencies of the task. When each task node performs all the tasks in turn, the complete signal is reported to the management server, and the results are transmitted to the storage center, the system then enters the reduce state. The system enters the finished state when all the tasks have been completed and all the results have been returned. The management server sends the GC command to each node that joins the computation, performs the garbage collection, releases resources, and waits until the next scheduling.

Task scheduling algorithm adopts the priority algorithm and the first come first serve (FCFS) hybrid scheduling algorithm, and adds the basic principles of the rotation method. In order to do this the MS Server maintains a task dictionary<int,Queue <Task>>. Where Key is the priority of the task queue, Queue <Task> is task queue, and Task is a single computing task. The algorithm principle is shown in Figure 4.

When the computational task is established, the system is statically assigned a priority value $K$, the $K$-value is between $1 \sim n$. The task enters the corresponding priority queue according to the $K$ value. The task is queued according to the first come first serve (FCFS) scheduling algorithm when it enters the queue because all tasks begin with the same priority. Viewed from a straight line, the algorithm is fair in a general sense, that is, each task will receive their turn when the previous task has completed and there are resources available. However, some tasks will take much longer to complete than others and some tasks with a short execution time may be delayed significantly if they arrive after these tasks, To compensate for this, the system uses the round robin method, and sets a time slice for each task. When the task has used the allocated time slice, the execution of the task is aborted, and the $K-1$ value of the task is determined. If the value of the $K-1$ is in the Dictionary Keys, that is, the value of Dictionary. ContainsKey $(K-1)$ is equal to true, then the task is removed from the head and added to the end of the Dictionary, it is contained in Dictionary $[K-1]$ team; otherwise it is added to the end of the Dictionary $[K]$ team. The choice of the time slice length will directly affect the system overhead and response time. If the time slice is too short the amount of taken to switch tasks will increase the overall cost of the system. If the time slice length is too long even the longest task will be executed within the time slice, the round robin algorithm will be ignored and the system will fall back to the FCFS algorithm. The selection of the time slice length can be determined according to the requirement of the response time of the system $R$ and the maximum allowable tasks number $N_{\max}$ in the queue, and it can be expressed as: $q = R/N_{\max}$. In the const value of $Q$, the response time of $R$ seems to be greatly reduced if the number of tasks in the queue is far less than $N_{\max}$. For system overhead purposes, the timing of task switching will not change due to the fixed value of $Q$. For simplicity, the system uses a fixed time slice.

The performance of task scheduling can be measured by parameters, such as task turnaround time, response time, throughput, and the utilization ratio of computing nodes. Here the focus is on the task turnaround time. The turnaround time for the task $i$ is defined as $T_i$, thus: $T_i = T_{ie} - T_{is}$. Where $T_{is}$ is the start time of the task and the $T_{ie}$ is the end time of the task completion. For $n(n >= 1)$ tasks, the average turnaround time is:

$$T = \frac{1}{n}\sum_{i=1}^{n} T_i$$

When the task is submitted to the system, it will be executed immediately until the task is Mapped, the task is then likely to enter the wait state. Setting $T_{iw}$ as the waiting time that the task waits between submission and Map, the correct turnaround time is then noted as $T_i$, and $T_i = T_{ir} + T_{iw}$, where $T_{ir}$ is the execution time. Furthermore, the weight of the turnaround time can be used to measure the scheduling performance. Define the weighted turnaround time as the ratio of task turnaround time to task execution time: $W_i = T_i/T_{ir}$.

For the $n$ tasks contained in the task flow, the average weighted turnaround time is:

$$W = \frac{1}{n}\sum_{i=1}^{n} W_i$$

## 4.2 Model evaluation

Through the revision and improvement of the scheduling algorithm in literature [12], the evaluation model of the system is as follows.

Assuming that the size of the particle is linearly related to the size of the task, the execution time $T_i$ is:

$$T_i = b_i + a_i x_i \tag{4}$$

$b_i$ is the time of initializing the system, $a_i$ is the task granularity linear growth factor, $x_i$ is the size of tasks.

Assuming that the data transmission time is linearly related to the size of the task, then,

$$\text{Data\_}T_{ij} = \text{Data\_}b_{ij} + \text{Data\_}a_{ij}x_i \tag{5}$$

In the formula, Data_$T_{ij}$ is the required time to transfer data from the task $i$ to the task $j$. Where Data_$b_{ij}$ is the time required to transmit the initialization data, Data_$a_{ij}$ is a linear factor.

Formulas (6)–(7) can be adopted to solve the *TCP* traffic model, referenced in the literature [14, 15]. In a high speed local area network with 100M/1000M adaptation, the ratio of the data transfer time and the computation time are small in the whole simulation process, that is because the transmission rate between computers is very high, while Data_$b_{ij}$ and Data_$a_{ij}$ are relatively small and a copy of the original data has been saved in the computing unit prior to the start of the calculation.

$$T(t_{RTT}, s, p) = \frac{c \times s}{t_{RTT} \times \sqrt{p}} \tag{6}$$

$$T(t_{RTT}, s, p) = \min\left(\frac{w_m \times s}{t_{RTT}}, \frac{s}{t_{RTT} \times \sqrt{\frac{2bp}{3}} + t_{RTo}\min\left(1, 3, \sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}\right) \tag{7}$$

For the 2 layer $m$ fork tree $DAG_i$, the size of the sub task is total $M$ copies, but the granularity of the $M$ subtasks are different. The relationship of the sub task scale is then $x_i = x_{1i} + x_{2i} + \cdots + x_{mi}$, assuming that the time of task execution and the task scale are a linear relationship, and the execution time of each subtask is $sub\_T_{ki} = b_i + a_i x_{ki}$. The data transfer time between the leaf node and the root node is $subData\_T_{ki} = Data\_b_{ij} + Data\_a_{ij}x_{ij}$, where $k = 1, 2, 3, \ldots M$. For task $i$, if it is not decomposed, the task completion time is calculated by the formula (1); if it is decomposed, then the formula (5) is used

$$T_i = \text{Max}(\text{sub}_{T_{1i}}, \ldots, \text{sub}_{T_{mi}}) + \text{sub\_root}_i \tag{8}$$

According to the characteristics of sub task diversity, the primary role of the root task is: transmit data from the root node to the leaf node, then compute and collect results from the leaf nodes to the root node, and transform the root task computing result to the *DAG* map of lower sub task. Therefore, the computing time of sub_root$_i$ is mainly the data transmission time. If the scheduling algorithm supports a data parallel transmission, sub_root$_i$ can be approximated by the formula (9):

$$\begin{aligned}\text{sub\_root}_i = {} & \text{Max}(\text{subData}_{T_{1k}}, \ldots, \text{subData}_{T_{mk}}) \\ & + \text{Max}(\text{Data}_{T_{i1}}, \ldots, \text{Data\_}T_{is})\end{aligned} \tag{9}$$

# 5. COMPUTE NODE ASSIGNMENT

MS loads the subtasks into a task list List<T> by reading the task description. Then, the task priority of each sub task is determined according to the dependency set list List<R> of each task in their description. In the calculation of node allocation, at first each of the sub tasks in the List<T> will be distributed into a different Dictionary<int, Queue<T>> according to the level of each sub tasks. Where the int is the task queue level, the Queue<T> is the same level task queue. The tasks in the queue are scheduled according to the FIFO strategy, and the high level sub task queue is given priority to compute the node assignment. The FIFO strategy is used to compute node allocation between tasks and subtasks. When the time slice of the task $T$ in the queue is expended, it will release the computing node, and then return to the end of the queue, waiting for rescheduling. When the task interdependence leads to competition for resources, the task will be sent to the low level queue by reducing the level of sub tasks, and this can solve the problem of deadlock caused by task preemption. The computing node is released and the system task is completed when the task is completed. The node will request to reassign the task and modify the state of the task in the MS. The MS will notify the subtasks that are waiting for the dependency to continue execution by event method.

# 6. COMPUTING DATA COMMUNICATION MODEL

According to the calculation model of the above design, master-slave mode and P2P mode are adopted for the communication and data exchange between the nodes, the chart of Compute node LC$_n$ startup flow as shown in Figure 5:

The node will run the joint computing program, which logs on when it is started. After the program starts, it firstly initializes the parameter information of the node. The service address of the managed server MS is stored in each LC compute node when the LC is remotely deployed by the system configuration. Using this parameter, each LC can detect the presence of the server and try to connect to it. If the connection is not successful, then the system link fails, the node cannot access the collaborative computing system, and it will become a calculation of ac-node. If the node can connect to the server, it will be registered on the MS, and the registration information will contain the basic information

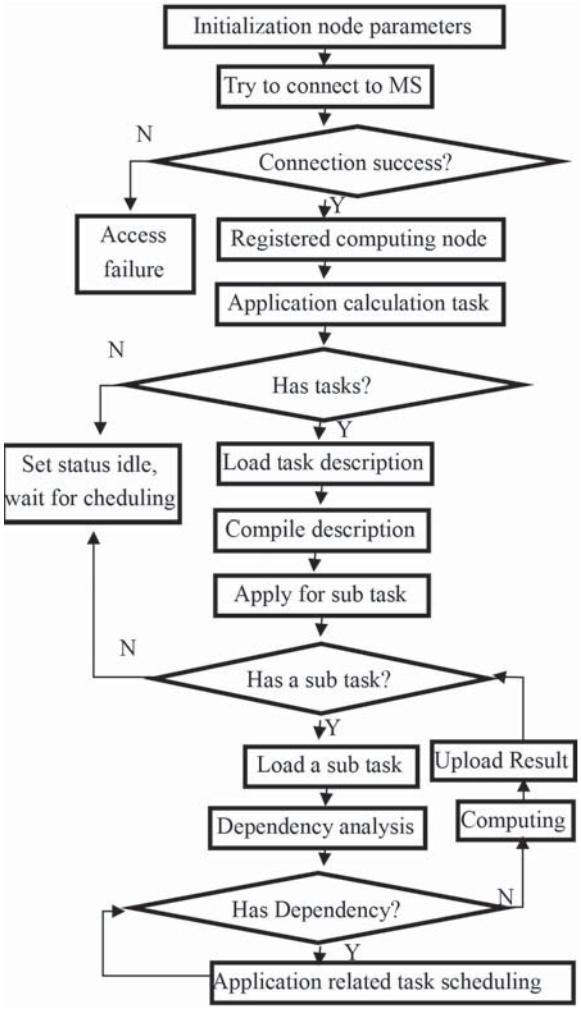


**Figure 5** Compute Node $LC_n$ Startup Flow.

of the node, computing power, etc.; LC can apply to the MS server to run programs in computing task nodes after successful registration. If the MS server does not have a task at this time, that is to say, if the federal computing system is idle, the node will then set itself to idle, waiting for scheduling. When the MS server has a RC application task it will scan the status of the local compute node client LC after it completes the initialization of the task. If the number of idle computing nodes LC the MS has scanned is more than 0, then it proceeds to resource allocation and task scheduling, if the number of idle nodes the MS scanned is 0, the task will be set into the task scheduling queue and wait due to the lack of resources. The idle LC will download the task specification and load it when it receives the signal from the MS scheduler. LC compiles the subtask execution code in task specification through a dynamic compilation system, and applies for the issuance of subtasks from MS after the task specification was compiled. Under normal circumstances, the subtask execution code in the task specification can be compiled through the instructions received from the MS. This can show that the calculation of the computing power of the node cannot meet the requirements of the task description if it cannot be compiled by the instructions. When the LC receives the subtask from MS, it carries out the task loading, and analyzes whether there are other subtask dependencies; if there is a dependency, the output parameters of the subtasks associated with this subtasks are first obtained. If LC can receive this data, it is illustrated that the subtask has been terminated and its output can be used as input parameters for this task, otherwise, the output data cannot meet the input of the task and the task is then required to recalculate the output in accordance with the requirements of this task. When the output of all dependent subtasks can satisfy the input of the task, the task is executed; the results of the calculation will be uploaded to the data sharing area for other subtasks. The LC that completed the task computing can be reinitiated and send a request to the MS for another subtask. If there are no subtasks available, LC is set to idle and waiting for the MS scheduler.

Three methods are used to realize the communication and data exchange between nodes. The data that has been calculated by the computing node and merged to the server can be applied by the other nodes that apply to the data management server. When the application for the identification of the identity of the consumer data is audited, the application node can consume data provided by the production data node; if the node is unable to meet the request of the data node to the management server, the reason for the failure of the data is checked; if the other computing node is

calculating the application data, the calculation node enters the wait state, and registers the waiting resource application to the MS server. When all the calculations are completed and all the results are reported to the data server, the MS server will find the waiting nodes from the resource application, and inform those application data nodes listed in the application form to load the data; if the data is not retrieved on the management server, and the current computing network does not have a computing node to compute the data, then the current computing node is set into the stack, and compute dependent data set.

In order to ensure the communication and event notification between the computing node and the MS server, and to mask the difference between the computing nodes hardware and heterogeneous structure of the operating systems, the computing nodes use the Net.Tcp communication protocol to provide remote services via an open Web service. In the system design, the Windows Communication Foundation (WCF) is adopted to provide data sources. The WCF provides a high performance network communication protocol based on the Net.Tcp protocol and system components with Net.Tcp Port Sharing Services, so that the port can be shared between multiple user processes. The data exchange uses the XML language which is based on the object transfer protocol, and this provides the possibility for the exchange of structured and solidified information between heterogeneous computing nodes. In addition, in order to ensure the data access security between nodes, the system uses a security algorithm based on the elliptic curve algorithm and federal verification [16].

Due to the different environment of LC and RC, the complexity of the RC host itself and the limitation of the access rights, the RC and LC systems are designed using different strategies. LC and RC also use different protocols in the communication method. LC uses the Net.Tcp protocol, but RC uses the pollingDuplexHttpBinding protocol. The HTTP protocol is commonly allowed through firewalls, using this protocol can prevent scheduling failures due to the RC node being blocked by a host firewall.

In order to make the management server simultaneously serve two kinds of protocols, it needs to be configured with netTcpBinding and pollingDuplexHttpBinding in the bindings section of the serviceModel section of the protocol. The pollingDuplexHttpBinding configuration is as follows:

```
<pollingDuplexHttpBinding>
    ......
</pollingDuplexHttpBinding>
```

This section is added into the <bindings> section. When configuring the netTcpBinding protocol, you need to add the following section to the <bindings> section:

```
<netTcpBinding>
    ......
</netTcpBinding>
```

The system uses the Silverlight rich client as the development model of *RC* in the *RC* endpoint. Silverlight does not support the WCF Security model; therefore to call this service in *SL*, the Security Mode must be set to None. By default, the Security Mode is Transport, so this section must not be omitted and must be explicitly configured.

When configuring the information about the service, two endpoint points need be added because of the adoption of the two protocols. There are two kinds of endpoints in the <services> node, one is called by the client, and the other is the publication of metadata for the generation of service information. Using <endpoint contract="IMetadataExchange" binding="mexTcpBinding" address="mex"/> node to publish metadata. Using <endpoint address="ForWinform" contract="NetTcpDuplexCommunication.Server.IService1" binding="netTcpBinding" bindingConfiguration= "tcpConfig"/> node to Configure client Net.TCP calls. Using <endpoint address="ForSilverLight" binding="pollingDuplexHttpBinding" bindingConfiguration = "pollingDuplexHttpBinding1" contract="EndoscopeIMS. Server.IServiceForEndoscopeCDS"/> node to Configure client pollingDuplexHttpBinding calls.

Silverlight can only use ports between 4502 and 4535 so it is imperative when adding baseAddresss to the host sectionto use a port of 4502.

When adding references it is important to write the whole address completely:

net.tcp://localhost:4502/NetTcpDuplexCommunication. Server/Service1.svc/ForWinform

In addition, the aspNetCompatibilityEnableds must be set to true, such as: <serviceHostingEnvironment multipleSiteBindingsEnabled="true" aspNetCompatibilityEnabled="true"/>

In order to complete the communication between the server and the client, the interface must be specified in the WCF interface. When a callback [ServiceContract(CallbackContract = typeof(IClientCallback))] attribute for the WCF interface is added, the [OperationContract(IsOneWay = true)] attribute must be added to all callback methods for the interface IClientCallback.

A method is included in the IService1 interface to register a subscribe server. The server call back *RC* or *LC* client by the registration of computing nodes in Register, the client callback, and the client to unsubscribe by UnRegister. Custom callback method PumpMessageModel is the implementation of callback push message pump. When the server callback to the client fails, the server will determine that the client is offline and that it must be removed from the list of server computing nodes.

On the server side, the [AspNetCompatibilityRequirements (RequirementsMode=AspNetCompatibilityRequirements Mode.Required)] and [ServiceBehavior(Concurrency Mode=ConcurrencyMode.Multiple,InstanceContextMode =InstanceContextMode.Single)] attributes must be added to the Service1 implementation class that inherits the IService1 interface. There is a static Dictionary<string, IClientCallback> _clients dictionary to save the registered remote computing node in the Class, and it is a static list of clients. The implementation of the Register is to add the current channel directly to the _clients dictionary for the invocation.

When the specified channel does not callback, the channel is removed from the _clients dictionary, and the computing

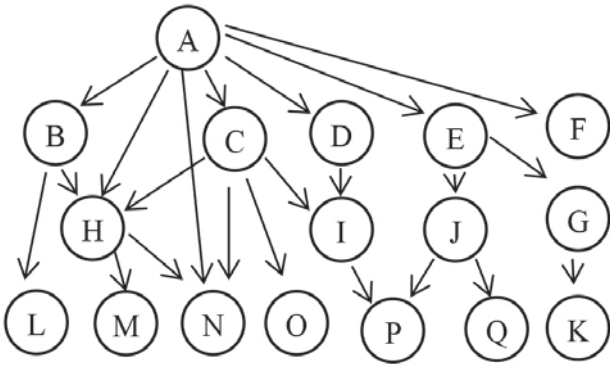


**Figure 6** The Dependency Set $R_i$ Relation.

node is declared dead. The node will no longer be assigned sub tasks and scheduled. The server will reclaim the task that has been assigned to the node, and then re-perform the Map on the other idle nodes.

When the system is deployed, as the WCF host uses a high version of IIS that supports net.tcp binding, it is necessary to enable the HTTP and the Net.Tcp protocols, and modify the net.tcp configuration to bind to 4502:*. Since WCF Activation is an optional component of Windows, it is not installed by default, so WCF Activation needs to be installed for the IIS support of WCF calls to non HTTP pipes.

In order to ensure that the RC of the Silverlight allows crossing domain access applications in the absence of policy files, cross domain configuration clientaccesspolicy.xml files are added to the WCF host publishing directory. In addition, the built-in program must have the trust level elevated to trusted in OOB mode.

## 7. USE CASE TEST OF COMPUTING MODEL

Given a computing cluster system $C$ which consists of Management server $M$, Storage service cluster $S_i$, and Computing node $N_i$, Then $C$ can be described as: $C = \{M, S_i, N_i\}$.

Given $i = 1$ of use case $C$ test calculation cluster $S_i$. $N_i$ is a collection of $\{N_1, N_2, N_3, N_4, N_5, N_{10}, N_{20}, N_{40}, N_{80}, N_{120}\}$.

Given the job $J$, the $J$ can be broken down into subtasks set $T_i$ and subtask dependency set $R_i$. Then $J$ can be described as: $J = \{T_i, R_i\}$.

Given the subtask set $T_i$ and the subtask dependency $R_i$ of the test case $J$, it can be described as: $T_i = \{T_A, T_B, T_C, T_D, T_E, T_F, T_G, T_H, T_i, T_J, T_K, T_L, T_M, T_N, T_O, T_P, T_Q\}$; $R_i = \{R_A \rightarrow R_{BCDEFHN}, R_B \rightarrow R_{HL}, R_C \rightarrow R_{HINO}, R_D \rightarrow R_i, R_E \rightarrow R_{GJ}, RG \rightarrow R_K, R_H \rightarrow R_{MN}, R_I \rightarrow R_P, R_J \rightarrow R_{PQ}\}$. The dependency set $R_i$ relation is shown in Figure 6:

By the dependency set $R_i$, the priority queue $Q_i$ in $M$ can be defined: $Q_i = \{Q_1, Q_2, Q_3, Q_4\} = \{\{A\}, \{BCDEF\}, \{HIJG\}, \{LMNOPQK\}\}$, here the queue priority is as follows: $Q_1 > Q_2 > Q_3 > Q_4$.

For the decomposed subtask $T_i$, its execution time can be described by a four tuple($T_{\text{in}}$, $T_{\text{out}}$, $T_{\text{instructions}}$, $T_{\text{commcapacity}}$), where $T_{\text{in}}$ is the time required to execute the task execution, which is dependent on the functional dependencies of the dependency set $R_i$ and the input data size; $T_{\text{out}}$ is the result of the output of the task to the data center, which is mainly affected by the output data scale and network communication ability; $T_{\text{instructions}}$ is the time required to compute the node $N_i$ execution of the subtask $T_i$, whose length is determined by the computing power of the node $N_i$ (the total number of instructions executed per second) and the total number of subtasks. The $T_{\text{commcapacity}}$ is a main expression of measuring the communication capacity of the node, where the communication throughput of the node $N_i$ is greater, and the time of each communication is shorter. The task simulation test case data are shown in Table 1:

Test results are shown in Figure 7 for the use case:

As can be seen from the graph, with the increase of computing nodes, the time required for the task is gradually reduced. However, when the number of nodes reaches a certain threshold, the time required by the number of nodes is gradually weakened. The main factor that affects the time required to complete the task is the computing capacity of a single node and the network communication speed, when the computing capacity is stronger and the communication speed faster, the less time is required.

## 8. CONCLUSION

This federal synergy computing model which the system provided with heterogeneous and dynamic characteristics can be applied to large-scale networks and supports the dynamic check in and check out of nodes. Using computer networks to connect heterogeneous computer devices to provide high performance computing capabilities is currently the most common method of super-large scale computing. With the help of previous research results, this paper proposes a compact scalable hybrid federal computing model based on literature [17–19]. To compare with the current mainstream network computing model, the implementation of the proposed method shields the differences of computer nodes in the software and hardware by the design of the application network protocol layer. Any computing device can access the system at any time to participate in the operation. This greatly reduces the cost of computing equipment and the formation of a network of inexpensive computing power and provides an alternative solution for the rapid implementation of a large scale computing network. The system has high expandability

**Table 1** Simulation Test Case Data.

| Task Name | TotalTask Instruction | Output DataSize | Network Communication Capability |
|---|---|---|---|
| **A** | 6128701 | 23552 | 921 |
| **B** | 5243421 | 22528 | 614 |
| **C** | 8336538 | 44032 | 204 |
| **D** | 4481950 | 46080 | 716 |
| **E** | 7788828 | 29696 | 716 |
| **F** | 9793721 | 47104 | 716 |
| **G** | 6797833 | 49152 | 716 |
| **H** | 6534583 | 16384 | 921 |
| **I** | 9688247 | 11264 | 102 |
| **J** | 2105543 | 46080 | 102 |
| **K** | 7359003 | 46080 | 716 |
| **L** | 1510364 | 38912 | 512 |
| **M** | 1215425 | 49152 | 819 |
| **N** | 9784983 | 26624 | 307 |
| **O** | 2083561 | 25600 | 512 |
| **P** | 1855908 | 32768 | 102 |
| **Q** | 5329746 | 27648 | 614 |

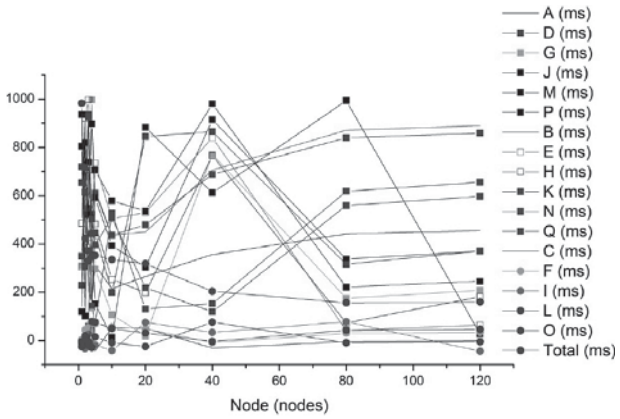


**Figure 7** Use Case Test Results.

and feasibility when compared with the method provided by literature [18]. The task decomposition algorithm in this paper is a further extension of the method mentioned in the literature [12], and further improves the application environment of the method. However, the task decomposition algorithm in this system cannot be completely decomposed by MS. This paper will focus on enhancing the automation and intelligence of the program and improving the task diversity algorithm. The calculation model proposed in this paper, to a certain extent, has the advanced nature and reference to solve this kind of method, and has certain practical significance for engineering guidance.

## ACKNOWLEDGEMENT

The study was supported by science and technology plan project of Henan province (No. 212102210395), and high level talent introduction research start project of North China University of Water Resources and Electric Power (No. 40427).

## REFERENCES

1. Szymanski, T. (2000). High Performance Computing with Optical Interconnects. *Proceedings of SPIE — The International Society for Optical Engineering*, p. 217–225.
2. Aoki, K., Yamagiwa, S., & Ferreira, K. (2004). Maestro2: High Speed Network Technology for High Performance Computing. *Proc of IEEE International Conference on Communications*, NJ: IEEE, p. 1033–1037.
3. Walker, E. (2007). Creating Private Network Overlays for High Performance Scientific Computing. *Lecture Notes in Computer Science*, p. 204–222.
4. Nishi, H., Tasho, K., Yamamoto, J., et al. (2000). A Local Area System Network RHiNET-1: A Network for High Performance Parallel Computing. *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*, p. 296–297.
5. Pronk, S., Pouya, I., & Lundborg, M. (2015). Molecular Simulation Workflows as Parallel Algorithms: The Execution Engine of Copernicus, a Distributed High-Performance Computing Platform. *J Chem Theory Comput, 11*(6), 2600–2608.
6. Calabrese, B., & Cannataro, M. (2015). Cloud Computing in Healthcare and Biomedicine. *Scalable Computing, 16*(1), 1–18.

7. Bezbradica, M., Crane, M., & Ruskin, H.J. (2016). Applications of High Performance Algorithms to large Scale Cellular Automata Frameworks Used in Pharmaceutical Modeling. *J Cell Autom, 11*(1), 21–45.
8. Somasundaram, T.S., & Govindarajan, K. (2014). CLOUDRB: A Framework for Scheduling and Managing High-Performance Computing (HPC) Applications in Science Cloud. *Future Gener Comp Sy., 34*(5), 47–65.
9. Liu, B., (2006). Agent Task Decomposition and Scheduling in Distributed Network Management. Nanjing: Southeast University.
10. Zheng, K. (2010). Operating System Concepts. *Translation*, Beijing: Higher Education Press, 11–13, 138–147.
11. Xiong, Kaiqi, & He, Yuxiong (2013). Power-efficent Resource Allocation in MapReduce Clusters. *Proceedings of the 2013 IFIP/IEEE International Symposium on Integrated Network Management*, p. 603–608.
12. Chen, Ming (2009). Distributed Computing Application Models. Beijing: Science Press.
13. Barnes, J., & Hut, P. (1986). A Hierarchical O(NlogN) Force-calculation Algorithm. *Nature, 324*(6096), 446–449.
14. Floyd, S., & Fall, K. (1999). Promoting the Use of End-to-end Congestion Control in the Internet. *IEEE/ACM Transcation on Networking, 7*(4), 458–472.
15. Padhye, J., Firoiu, V., Towsley, D., et al. (1998). Modeling TCP Throughput: A Simple Model and its Empirical Validation. In: Oran, D., ed. *Proceeding of the SIGCOMM*. Vancouver: ACM Press, 303–314.
16. Chen, H., Wang, H., Li, Y., et al. (2014). Security Token and Federated Authentication Based on Elliptic Curve Groups Algorithm. *Microelectron Comput, 31*(11), 88–91.
17. Javadi, B., Abawajy, J.H., & Akbari, M.K. (2008). Performance Modeling and Analysis of Heterogeneous Meta-computing Systems Interconnection Networks. *Comput Electr Eng, 34*(6), 488–502.
18. Yuan, L. (2011). A Heterogeneous Wireless Network Interconnection Strategy Based on IP Switching. *Proc of the 2011 IEEE International Conference on Computer Science and Automation Engineering, CSAE, 4*, 474–477.
19. Javadi, B., Akbari, M.K., Abawajy, J.H., et al. (2009). Multi-cluster Computing Interconnection Network Performance Modeling and Analysis. *Future Gener Comp Sy, 25*(7), 737–746.