Distributed Cryptographic Computing on Grid

Zhonghua Jiang[12], Dongdai Lin[1], Lin Xu[12], and Lei Lin[12]
(Corresponding author: Zhonghua Jiang)

State Key Laboratory Information Security, Institute of Software, Chinese Academy of Sciences[1]
Beijing 100080, P.R.China (Email: {jzh, ddlin, xulin, rockylin}@is.iscas.ac.cn)
Graduate School of the Chinese Academy of Sciences, Beijing 100039, P.R.China[2]

(Received Oct. 26, 2005; revised and accepted Dec. 10, 2005)

Abstract

Distributed cryptographic computing system plays an important role in cryptographic research since cryptographic computing is extremely computation sensitive. There are many research results done in this aspect, but no general cryptographic computing environment is available for cryptographic researchers and engineers. Grid technology can give an efficient computational support for cryptographic applications. Therefor, we put forward a general grid-based computing environment called DisCrypto for distributed cryptographic computing. In this paper, we simply describe the architecture of DisCrypto at first. The policy of task division adapted in DisCrypto is then analyzed. The method to manage subtask is further discussed in detail. Furthermore, the building and execution process of an execution plan is revealed. Finally, the details of DisCrypto implementation under Globus Toolkit 4 are illustrated.

Keywords: Computational Grid, cryptography, distributed cryptographic computing, task dividing

1 Introduction

A general-purpose cryptographic computing system plays an important role in cryptographic research and becomes a hot topic step by step. In 1987, Lenstra [10] developed a distributed factoring system which made use of electronic mail for the distribution of the programs and for inter-processor communication. Several 100 digit integers were factored within one month over it. In 1999, Selkirk [13] developed a system to solve ECCp-97 (a 109-bit elliptic curve challenge). The system connected over 1200 machines from at least 16 countries solved the challenge within 53 days. These machines communicated with TCP/IP connection or electronic mail. In 2000, Asbrink [1] developed another factoring system, in which Quadratic Sieve algorithm [2] is implemented and the parallelism is achieved using the Message Passing Interface (MPI) [9]. Due to the complexity of cryptographic computing and insufficient technology, all of these systems are not general-purpose but special to solve one kind of cryptographic computations.

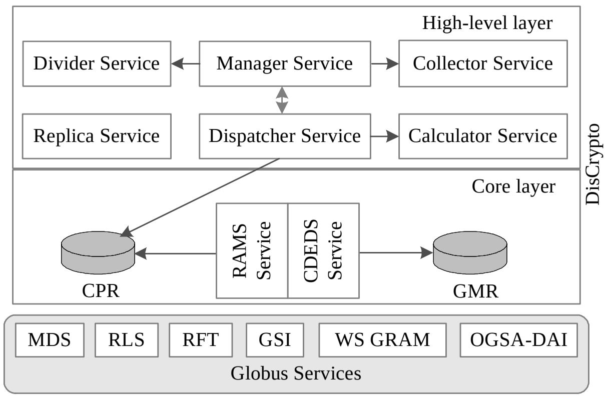

Figure 1: DisCrypto architecture

Computational grid is an infrastructure that enables the integrated use of remote high-end computers, databases, scientific instruments, networks and other resources by communities (“virtual organizations”) [5, 8]. Similar to electric power grid, computational grid can supply end users with general computational power which is geographically and device independent. Grid has been widely applied to many fields, such as highenergy physics, material science, business and engineering applications. Essentially all major grid projects are currently built on protocols and services provided by the Globus Toolkit [4]. New-generation grid technologies are evolving toward an Open Grid Services Architecture (OGSA) [6, 14] and Web Services Resource Framework (WSRF) [7] in which a grid provides an extensible set of services that virtual organizations can aggregate in various ways. The Globus Toolkit 4.0 (GT4) is based on the OGSA and WSRF concepts. However, to our best knowledge, till today, no relevant effort has been devoted to applying Grid techniques to perform efficient cryptographic computing in large distributed computing sites.

We have initiated a research effort to design and implement a novel infrastructure called DisCrypto for generally cryptographic computations on grid. DisCrypto is a general, open, scalable and high-performance cryptographic computing environment. We organize this paper as follows. The architecture and its main components is outlined in Section 2. The policy for task division is discussed in Section 3. The management of subtask dividing and collecting is introduced in Section 4. The building and execution process of an execution plan is revealed in Section 5. The implementation details under GT4 are introduced in Section 6. We briefly conclude the paper in Section 7.

2 Architecture

DisCrypto, defined on top of grid toolkits and services, aims to conduct cryptographic computing. So it should be general-purpose, open, scalable and high-performance. There concerns problems in DisCrypto including task-division, result-collection, task-calculation, execution management, fault-tolerant and resource management (computational engines, code and data), which are solved by the collaboration of several basic and special grid services.

2.1 Grid Services

The grid services of DisCrypto are special to cryptographic computing, which are organized in two layers: core layer and high-level layer. The former refers to services directly implemented on the top of generic grid services and takes charge of resource management, the latter is used to describe, control and execute parallel and distributed cryptographic computational task, and also offers services to conveniently replicate datasets and codes among grid nodes, which services are Replica, Divider, Collector, Manager, Dispatcher and Calculator.

Figure 1 shows layers as implemented on the top of Globus services as well as the grid data and metadata repositories.

2.2 Core Layer

The main goals of the Core layer are the management of all metadata which describe the features of data sources (large datasets), third-party computing engines (tools), and cryptographic computing codes. Moreover, this layer coordinates the application execution by attempting to fulfill the application requirements and the available grid resources. This layer comprises two main services.

CDEDS service extends the basic Globus MDS4 service. It is responsible for maintaining a description of all computational code, data and engines used in the DisCrypto. The metadata, managed by the CDEDS, include the following kinds of objects: computing tasks and corresponding results; computational code used to divide, collect and calculate subtasks; computational engines used to calculate subtasks.

The metadata are represented by XML documents and stored in a Metadata Repository (GMR). The CDEDS service is then used to search and access computational data, also to find previously registered codes and computing engines.

RAMS service is used to automatically find the best mapping between a dynamic execution plan and available resources. The Cryptographic Plan Repository backups these dynamic execution plan. The mapping of allocating resources has to be effectively obtained. After the execution plan activation, this layer manages and coordinates the execution of the computation.

2.3 High-level Layer

Divider and ***Collector*** *services* are used to automatically load and use dividing and collecting algorithms for task-subdividing and result-collecting. They both can serve multiple dividable subtasks at one time. A cryptographic job can be divided into many subtasks using task-subdividing algorithms.

Manager *service* manages (e.g., create, invoke and destroy) all the Divider and Collector instances.

Calculator *service* is responsible for atomic subtask's calculating. After accepting a calculation request, the Calculator parses the atomic subtask data, then decides proper tools or codes. If the site lacks corresponding Plug-in, the Calculator will invoke the Replica to replicate corresponding tools or code from some remote site. Then Calculator starts to calculate the atomic subtask, further invoking Dynamic Libraries by Java JNI or start computing engine by GT4 WS-GRAM service. Lastly, it returns the calculation result to the source Dispatcher.

Replica *service* is used to replicate implemented algorithms, engines or large datasets between a remote site and the local system. This service is based on the core CDEDS service. On the basis of the requirements and constraints of the Dispatcher service, the Replica automates the searching and finding of objects and securely transferring them by RFT service, which is included in GT4.

Dispatcher *service* functions as a subtask dispatcher and control the interactions among DisCrypto nodes. The core service RAMS decides proper nodes by their available resources and the resource demands of subtasks by execution plan.

3 Task Subdividing and Calculating

To achieve generality, flexibility and openness, DisCrypto requires a unified mechanism to divide, express, transfer subtasks and collect corresponding results of subtasks. A submitted job called *root task* can be divided into subtasks. These subtasks fall into two categories, *atomic* and *dividable*. Atomic task can't be further subdivided while a dividable task need be subdivided recursively by corresponding dividing algorithms at potentially different grid nodes. Through the recursive process of task division, a *task division tree* TDT finally comes into being, in which the root task is root, the dividable tasks are branches, and the atomic tasks are leaves. In our implementation, an XML document is used to describe a subtask.

Task-subdividing and result-collecting are two key concerns to perform distributed, especially general, cryptographic computing. It is impossible to design a generic dividing, collecting or calculating algorithm because of diversity of cryptographic computing problems.

An atomic subtask requires a task-calculation algorithm; a dividable subtask requires a task-division and a task-collect algorithm. For many computing tasks, the codes of their task-calculation, task-dividing or task-collecting algorithms have been available. To utilize existing algorithms and to make use of new algorithms, we extract the metadata of an implemented algorithm and represent in XML documents, and save to GMR. The metadata for a task-division algorithm with the implemented algorithm is call a Meta-divider. Similarly, Meta-collector and Meta-calculator are used for implemented task-collecting and task-calculating algorithms.

To divide a subtask, Divider instance must find a Meta-divider which should match the subtask's *problemName* attribute. The attribute decides the task type of subtask and Meta-divider. Lacking a Meta-divider, Manager service will invoke the Replica to search and duplicate one from some remote node. For a new computing problem, metadata of its Meta-divider, Meta-collector and Meta-calculator should be registered to GMR through the CD-EDS service.

4 Task Management

How to well manage subtasks dynamically created by divider instances is critical in a large scale cryptographic computing environment. There are two factors that must be considered. One is to achieve maximum parallel degree for a cryptographic computing job. The other is to avoid overload of a grid node due to too many subtasks which are being processed. In order to satisfy these requirements, we need make effort to control the growth of corresponding TDT.

A divider instance is called an *atomic divider* (D_a) if all subtasks divided by the divider are atomic. A divider instance is called a *dividable divider* (D_d) if all subtasks come from the divider are dividable. Note that we confine that all subtasks from D_a are parallel while these subtasks from D_d may be either parallel or sequential. If a dividable task can be divided into both parallel and sequential subtasks, more than one divider algorithm are required for the dividable task.

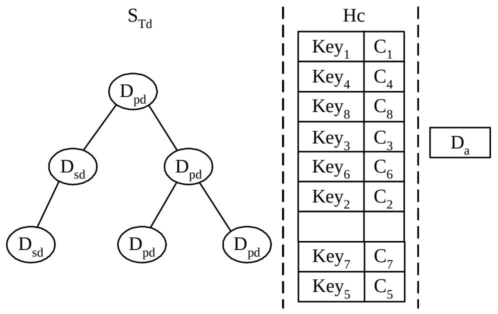

Figure 2: A structure for subtask management

```
Algorithm getDividableDivider
Input: S is the root of $S_{T_d}$
Output: A new subtask whose divider in $T_d$
  while(S!=NULL)
    if(S.divider.isAtomic()||S.getState()="FINISHED"
        ||S.getState()="WAIT")
      S ← S.InOrderNext()
    else
      return S.getNewSubtask()
  return NULL
```

Figure 3: Generating a subtask whose divider is a D_d

No matter what type of divider instance is used by a dividable task, we need't further classify its corresponding collector instance since we can easily manage all collectors for a job. During computing, these subtasks on the TDT are distributed over different grid nodes; all atomic tasks are maintained by calculator instances; dividable tasks are managed by manager services.

A hash table is called a *Collector Hash Table* (H_c) which stores all working collector instances for a job in a special grid node. In H_c, the key is a 2-tuple *(TaskType, TaskName)* of a dividable subtask, and the corresponding value refers to corresponding collector instance. To rapidly collect and assemble, all active collector instances are put into a H_c which is managed by the manager service of a grid node.

A tree is called a *Dividable Divider Tree* (T_d) if all working dividable divider instances $\{D_d\}$ for a job are hold in the tree. In general, the T_d of a job is distributed on multiple grid nodes where each node maintains a subtree of T_d. A subtree of T_d is denoted by S_{T_d}.

Now we discuss the working process of Manager service. In a grid node, Manager service maintains a 3-tuple *(S_{T_d}, H_c, D_a)* for a cryptographic computing job. Figure 2 shows an example of the structure. D_a is the only working atomic divider instance which creates atomic subtasks for Calculator Services. H_c is used to rapidly collect results of its subtasks. Newly created D_d for a dividable subtask is automatically added on S_{T_d}, and a finished D_d is removed dynamically from S_{T_d}. When a new divider instance is created and added on S_{T_d}, the corresponding

```
Algorithm getAtomicDivider
Input: S is the root of the subtree
Output: A new subtask whose divider is atomic
  while(S!=NULL)
    if(S.divider.isAtomic() && S.getState()="OK")
      return S.getNewSubtask()
    else if(S.getState()="FINISHED" or "WAIT")
      S ← S.InOrderNext()
    else
      task ← S.getNewSubtask()
      while(isDividable(task))
        divider ← new Divider(task)
        S.insert(divider)
        S ← divider
        H.put(new Collector(task))
        task ← S.getNewSubtask()
      return task
  return NULL
```

Figure 4: Generating a subtask whose divider is a D_a

```
Algorithm putSubresult
Input: key=(taskType, taskName, parentHash)
       result=taskResult
Output: A state or global result
  collector=H.get(Key)
  if(collector=NULL) return "FINISHED"
  state=collector.put(result)
  if(state="FINISHED")
    result=collector.getResult()
    H.remove(key)
    divider=S.search(key)
    if(divider!=NULL)
      for(each skey in S.traverseSubtree(divider))
        H.remove(skey)
      S.removeSubtree(divider)
    if(S=NULL) return result
  return "OK"
```

Figure 5: Assemble a subresult

collector instance is put into H_c at same time. When a D_d or D_a is completed, the D_d will be firstly removed from S_{T_d}, the D_d or D_a then will be destroyed. Note that when a D_d or D_a is destroyed, the corresponding collector instance is till working for subresult collecting. However, if the result of a dividable subtask is gotten, then the corresponding collector instance will removed from the H_c and destroyed; its corresponding dividable instances and all of its lower-level divider and collector instances will also be recursively removed and destroyed from S_{T_d}.

In Figure 2, D_{pd} denotes a parallel D_d while D_{sd} denotes a sequential D_d. In our implementation, a S_{T_d} is converted into a binary tree for convenience of traverse. To avoid expansion of S_{T_d}, we traverse a S_{T_d} in *inorder* when searching a D_d. Figure 3 shows how to get a subtask whose divider is a D_d; Figure 4 shows how to get a subtask

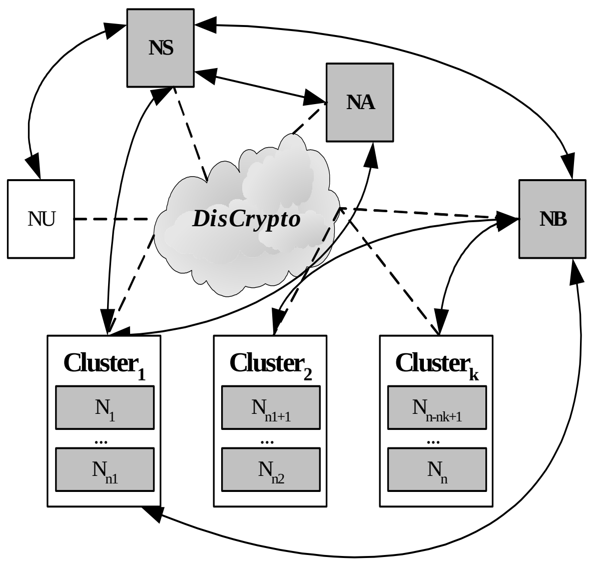

Figure 6: A cryptographic computing example

whose divider is a D_a; Figure 5 shows how to assemble a subresult. The method for subtask management shown in Figure 2 and corresponding algorithms shown in Figures 3, 4 and 5 assure that the parallel degree of subtasks is maximized and that the load of subtask management is minimized.

5 Task Building and Execution Process

We suppose that a grid user (GU) needs to perform cryptographic computing by corresponding implemented algorithms. The task could be performed as described in Figure 6.

Step 1: the *GU* starts the search of computational resources for executing the cryptographic computing task from his/her grid node (*NU*). The search, performed by means of the CDEDS, locates the computational resources needed to execute the computing process, respectively by the $Cluster_1$, $Cluster_2$, ..., $Cluster_k$ Globus Nodes, on which computing process will be performed in parallel. The search process compares the meta-data (about Meta-Dividers, Meta-Collectors, Meta-Calculators, memory, libraries, etc.) against the features of *Cluster*s and *NS*, *NA* and *NB* nodes.

Step 2: the *GU* builds an execution plan for the cryptographic computing task, specifying strategies for tools, code and code movement, and for algorithm execution. The execution plan is built by the Dispatcher and is stored into the local CPR.

Step 3: the *GU* starts the computation by submitting the executing plan to the RAMS. The following steps

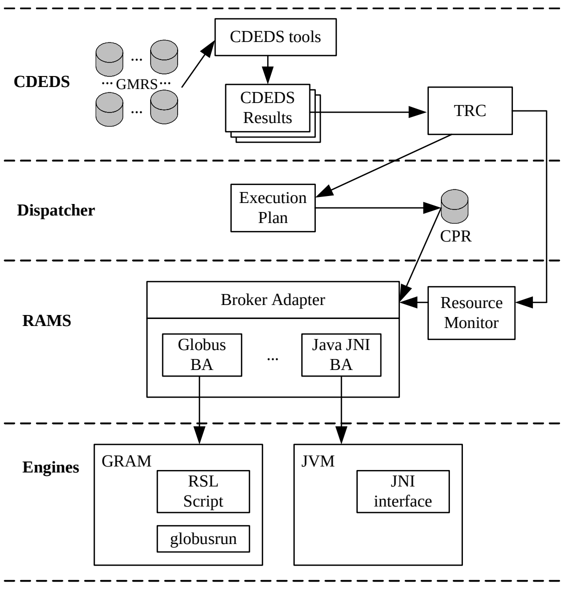

Figure 7: Task building and execution process

are then executed.

a. From NS, $Dividers$ and $Collector$s of the job are staged on NS, NA and NB, and Calculators is staged on each cluster node. The staging process (i.e., code movement and eventual installation on a target node) is executed by the Replica service on NS.

b. On NA, Dividers divide the job into subtasks T_1, ..., T_l. If $T_i \in T_a$, T_i will be moved into $Cluster_j$ to computing in parallel; while if $T_i \in T_d$, T_i will be moved into NS, NA or NB to conduct further division and collection.

c. NS, NA and NB collaborate on dividing dividable subtasks and collecting corresponding results. $Cluster_i$ calculates atomic subtasks in parallel till the job is finished.

Step 4: the final result of computation is transferred to NU and visualized. The result is also stored in CPR by the means of the local CDEDS.

The task building and execution process is shown in Figure 7. The Task Composer (TC), an internal Dispatcher module, assists the user in building the execution plan. It presents to the user a set of graphic objects represents of resources (e.g., Meta-dividers, Meta-collectors, Meta-calculators, computing engines and computational nodes), which metadata were previously stored into the Task Resource Cache (TRC). These metadata, containing information about resources selected to perform the computation, are extracted from local or remote CPRs as result of RAMS queries. Therefore, the user can compose these objects using common visual facilities (e.g., drag and drop), to form a graphic representation of the application data flow. The composition is validated its logical consistency, and hence its XML representation is generated, that is the execution plan.

```
<EPlan>
  ...
  <Stage label="replica1" type=" replica" >
   <From location="NS" MetaData=" dc_a.xml"  desc=" DesDiff DC on NS" />
   <To location="NA"  MetaData=" dc_aa.xml"  replicakind="optional"/>
  </Stage>
  <Stage label="replica2" type="replica" >
   <From location="NS" MetaData="dc_b.xml"  desc="DesExh DC on NS" />
   <To location="NB"  MetaData="dc_bb.xml"  replicakind="optional" />
  </Stage>
  <Stage label="replica3" type="replica" >
   <From location="NS"  MetaData="dc_c.xml"  desc="DesDiff Cal on NS" />
   <To location="HC1 HC2 HC3"  MetaData="dc_cc.xml"  replicakind="optional" />
  </Stage>
  <Stage label="replica4" type="replica" >
   <From location="NS" MetaData=" dc_d.xml"  desc="DesExh Cal on NS" />
   <To location="HC1 HC5 HC7"  MetaData="dc_dd.xml"  replicakind="upgrade" />
  </Stage>
  <Stage label="computation0"  type="computation" >
     <Program problem="DiffExhCombiner" type="divide collect"  desc="Root Task on NS"  />
     <Param InputRef="root.xml"  to="NA NB"  subtaskPrefix="descom" />
 </Stage>
  <Stage label="computation1"  type="computation" >
     <Program problem="DesDiff" type="divide collect"  desc="Dividing and Collecting on NA"  />
     <Param InputRef="NS:descom_a.xml"  to="C1 HC2 HC3"  subtaskPrefix="DesDiffrential" />
 </Stage>
  <Stage label="computation2"  type="computation">
     <Program problem="DesExh"  type="divide collect"  desc="Dividing and Collecting on NB"  />
     <Param InputRef="NS:descom_b.xml"  to="HC1 HC5 HC7"  subtaskPrefix="DesExhaust" />
 </Stage>
  <Stage label="computation3"  type="computation">
     <Program problem="DesDiff"  type="calculate"  desc="Calculting on HC1/2/3"  />
     <Param InputPrefixRef="NA:DesDiffrential"  to="NA"  subresultPrefix="DesDiffrentialRes" />
 </Stage>
  <Stage label="computation4" type="computation">
     <Program problem="DesExh"  type="calculate"  desc="Calculting on HC1/5/7"  />
     <Param InputPrefixRef="NB:DesExhaust"  to="NB"  subresultPrefix="DesexhaustRes" />
 </Stage>
 <Computation Startpoint="computation0">
      <StageLink From="replica1" To="computation1"/>
     <StageLink From="replica2" To="computation2"/>
     <StageLink From="replica3" To="computation3"/>
     <StageLink From="replica4" To="computation4"/>
 <Computation>
  ...
</EPlan>
```

Figure 8: A fragment of execution plan

As an example, the execution plan in Figure 8 describes the cryptographic computation at high level, not containing physical information about resources (which are identified by metadata references), nor about status and current availability of such resources.

In fact, specific information about the involved resources will be included in the next phase, when the execution plan is translated in the particular broker language. The label attribute for the cryptographic example described above. The execution plan gives a list of stages and stage links, which are specified using respectively the XML tags *Stage* and *StageLink*. The *label* attribute for Stage element identifies each basic Stage in the execution plan, and is used in linking various basic tasks to form the overall task flow. Each *Stage* element contains a stage-specific sub-element, which indicates the parameters of the particular represented stage. For instance, the stage identified by the *replica1* label contains a *Replica* element, indicating that it is an implemented algorithm replication task. The *From* and *To* elements specify source and destination positions of the transfer. The *MetaData* attributes of such elements specify the detail location of metadata about source and destination objects. In this example, metadata about source of the implemented algorithm transfer in the *replica1* stage are provided by the *dc-a.xml* file, whereas metadata about destination are provided by the *dc-aa.xml* file. The *dc-a.xml* document provides metadata about the Meta-divider and Meta-collector stored on NodeS, whereas the dc-aa.xml document provides metadata about the Meta-divider and Meta-collector when stored, after data transfer, on NA. Both *dc-a.xml* and *dc-aa.xml* files are stored in the local TRC. The *StageLink* elements represent the relations among the tasks of execution plan. For instance, the first *StageLink* specified in Figure 8 indicates that the stage flow proceeds from the stage *replica1* to the stage *computation1*, as specified by its *From* and *To* attributes.

The Broker Adapter (BA) maps the generic execution plan, represented by an XML document, into a specified Globus RSL script, which can be directly executed by means of the *globusrun* program. The mapping process performed by BA is based on the Resource Monitor (RM) tool, which provides physical information of resources referenced in the execution plan. Static information (e.g., CPU speed, memory size, etc.) is provided by the RM directly accessing metadata stored in the CDEDS (a local cache in DisCrypto), while dynamic information (e.g., current CPU load and network latency) is provided by means of specified testing tools, for instance based on standard Unix or TCP/IP utilities.

6 Implementation

As discussed in Section 2, the architecture is composed of two hierarchic layers: the Core layer and the High-level layer. To deploy cryptographic computing applications we have implemented the High-level services (Divider, Collector, Manager, Replica, and Dispatcher Grid services) which are useful to start cryptographic computing Grid application. Moreover, we have also implemented the former layer on top of the GT4 services, which include CDEDS, RAMS and some basic tools allowing a user to publish metadata of the cryptographic computing Grid objects (Meta-divider, Meta-collector, Meta-calculator and computing engines).

6.1 Metadata Management

Each grid node declares the availability of DisCrypto objects (resources, components and services) by publishing specific entries into the Directory Information Tree (DIT) maintained by a LDAP server such as Grid Resources Information Services (GRIS) provided by Globus Toolkit.

Metadata are implemented by XML documents, on the basis of a set of specific information for the discovery and the use of resources. For instance, metadata about cryptographic computing tools provide information about the implemented task (e.g., dividing, collecting and calculating), complexity of the used algorithm, location of executables and manuals, syntax for running the program, format of input data and results, etc.

6.2 CDEDS Service

The discovery of interesting resources over the grid is accomplished in two steps: the metadata repositories are first located, searching LDAP directories for specific entries. Then the relevant XML metadata are downloaded from the specified GMR and stored in the local GMR or directly in the local file system. The collected XML metadata are then analyzed to find more specific information. We implemented the basic tools to find, retrieve and select metadata about grid resources (e.g., computational data, implemented algorithms and computational engines), on the basis of different search parameters. The useful metadata are then stored in the TRC, a local cache which contains information about resources (nodes, algorithms) selected to perform a computation.

6.3 RAMS Service

This component of the Core layer is responsible to start and manage the execution of a cryptographic computing task. Currently, we implemented the Broker Adapter that maps the execution plan, represented by an XML document, into a specific Globus RSL script, which can be directly executed by means of the globusrun program. In this way the DisCrypto application can be started as a Globus application by using its basic allocation services. The RAMS module will be completed by implementing dynamic monitoring and management functions for submitted applications.

6.4 Implemented Algorithms

For test purpose, distributed DES Exhaustive Key Search algorithm [12] (Ex-DES), Parallel Collision Search algorithm [11] (PCS) for elliptic curve and DES Differential Cryptanalysis algorithm [12, 3] (Dif-DES) have been implemented and integrated into DisCrypto. We separately implements three algorithms with size of 56KB, 51KB and 53KB for dividing, collecting and calculating of Ex-DES. Dif-DES includes two stage, differential analysis and exhaust search. So we implement three algorithms for dividing and collecting and two algorithms for calculating; the largest size of them is 58KB; the size of task data is about 2.3MB. PCS has also three algorithms for dividing, collecting and calculating, of which the largest size is 65KB.

We have carried out several experiments in LAN environment, which utilized these implemented cryptographic algorithms (i.e., Meta-dividers, Meta-collectors and Meta-calculators). Results show the speedups are nearly linear as the number of grid nodes increases.

7 Conclusion

The Grid infrastructure is growing up very quickly and is going to be more and more complete and complex both in the number of tools and in the variety of supported applications. DisCrypto is a kind of grid application dedicated for cryptographic computing. A general task-dividing mechanism is given. A management mode for subtask management maximizes parallel degree and minimizes load of subtask management. An execution plan for a job will be mapped to the most proper grid resources by RAMS service. In short, DisCrypto is a grid based cryptographic environment, which is general-purpose, open, scalable and high-performance.

Acknowledgement

This paper is supported by the National Grand Fundamental Research 973 Program of China (No. 2004CB318004), the National Natural Science Foundation of China (NSFC90204016), and the National High Technology Development Program of China under Grant (863, No. 2003AA144030).

References

[1] O. Asbrink, and J. Brynielsson, *Factoring Large Integers using Parallel Quadratic Sieve*, Royal Institute of technology, Sweden, 2000.

[2] D. Atkins, M. Graft, A. K. Lenstra, and P. C. Leyland, "The magic words are squeamish ossifrage," in *Advances in Cryptography – ASIACRYPT'94*, LNCS 917, pp. 263-277, Springer-Verlag, 1995.

[3] E. Biham, and A. Shamir, "Differential cryptanalysis of the full 16-round DES," *Advances in Cryptography – CRYPRO'92*, LNCS 740, pp. 487-496, Springer-Verlag, 1993.

[4] I. Foster, and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal Supercomputer Application*, vol. 11, no. 2, pp. 115-128, 1997.

[5] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: enabling scalable virtual organization," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200-222, 2001.

[6] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid services for distributed system integration," *IEEE Computer*, vol. 35, no. 6, pp. 37-46, 2002.

[7] I. Foster, J. Frey, S. Graham, et.al., *Modeling Stateful Resources with Web Services Version 1.1.* Global Grid Forum Draft Recommendation, 2004.

[8] I. Foster, and C. Kesselman, *The grid 2: Blueprint for a New Computing Infrastructure*, USA: Elsevier Inc, 2004.

[9] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, Scientific and engineering computation, Cambridge: MIT Press, USA, 1994.

[10] A. Lenstra, and M. Manasse, "Factoring by electronic mail," in *Advances in Cryptography – EUROCRYPT'89*, LNCS 434, pp. 355-371, Springer-Verlag, 1990.

[11] P. C. V. Oorschot, and M. J. Wiener, "Parallel collision search with cryptanalytic applications," *Journal of Cryptology*, vol. 12, pp. 1-28, 1999.

[12] B. Schneier, *Applied Cryptography.* John Wiley & Sons, Inc., 2nd edition, 1996.

[13] A. P. L. Selkirk, and A. E. Escott, "Distributed computing attacks on cryptographic systems," *BT Technology Journal*, vol. 17, no. 2, pp. 69-73, 1999.

[14] S. Tuecke, K. Czajkowski, I. Foster, et al., *Open Grid Services Infrastructure (OGSI) Version 1.0*, Global Grid Forum Draft Recommendation, 2003.

Zhonghua Jiang received his M.E. degree in Computer Science from Xi'an University of Technology (P.R. China) in 2003. Currently he is a Ph.D. candidate at the State Key Laboratory of Information Security, Institute of Software of the Chinese Academy of Sciences. His research interests include cryptography, networks security and grid computing. E-mail address: jzh@is.iscas.ac.cn.

Dongdai Lin is a full time research professor and deputy director of State Key Laboratory of Information Security, Institute of Software of the Chinese Academy of Sciences. He received his B.S. degree in mathematics from Shandong University in 1984, and the M.S. degree and Ph. D degree in coding theory and cryptology at Institute of Systems Science of the Chinese Academy of Sciences in 1987 and 1990 respectively. His current research interests include cryptology, information security, grid computing, mathematics mechanization and symbolic computations. E-mail address: ddlin@is.iscas.ac.cn.

Lin Xu received his B.S. degree in Computer Science in 2003 from University of Science and Technology of China(P.R. China). Currently he is a Ph.D. candidate at the State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences. His research interests include theoretical cryptography, provable security and cryptography algorithms. E-mail address:

xulin@is.iscas.ac.cn.

Lei Lin received his Bachelor degree in School of Software Engineering in University of Science and Technology of China (USTC) in 2004. and currently (2005) a Master candidate at the State Key Laboratory of Information Security(SKLOIS), Institute of Software of the Chinese Academy of Sciences. His interests are Networks Security, Cryptography, Distributed Computing, Bible Study and football. Email address: rockylin@is.iscas.ac.cn.