IJCSC International Journal of Computer Science & Communication Vol. 1, No. 2, July-December 2010, pp. 259-265

Grid-Bank for Distributed Systems Sharing and Integration: A Case Study

Tarun Kumar Ghosh[1] & **Parna Chatterjee**[2]

[1,2]Department of Computer Science & Engineering, Haldia Institute of Technology, West Bengal, India
Email: [1]tarun_ghosh_2000@rediffmail.com, [2]parna.chatterjee@gmail.com

ABSTRACT

A new emergent infrastructure for Internet-based parallel and distributed computing is computational Grids. This is a large scale, heterogeneous collection of autonomous systems, geographically distributed and interconnected by heterogeneous networks. Job sharing (computational burden) is one of the major difficult tasks in a computational grid environment. Many Grid projects have developed technologies that provide many of the services such as security, uniform access, resource management, scheduling, application composition, computational economy, with an exception of accountability. A new infrastructure called GridBank provides services for accounting to overcome this limitation [1]. We have simulated the banking system (accounting and payment services) that provide the information about the consumed resources by the system and also provide the mechanisms for service providers to be paid for authorized use of their resources. We have also simulated the banking system for recording of usage data, secure storage of that data, analysis of that data for purposes of billing and so forth.

Keywords: Computational Grid, Grid Accounting, GridBank, Payment Strategies.

1. INTRODUCTION

Grids or grid computing systems enable the integration of high-speed computer systems such as super-computers and clusters, networks, databases and other resources owned and managed by multiple organizations. Grids often span multiple networks and have to provide secure resource sharing across organizational boundaries. The vision is a global computing power Grid where computing power is as accessible as the electricity provided by the electric power grid.

Computational Grids are emerging as a new infrastructure for Internet-based parallel and distributed computing. In this system resources can be shared, exchanged, discovered, aggregated and distributed across multiple administrative domains, organizations and enterprises. For accomplishing this, Grids need an infrastructure which supports various services: security, uniform access, resource management, scheduling, application composition, computational economy, and accounting. In Grid banking system, resources are interconnected via the Internet with middleware, which is supporting remote execution of applications constitute what is called *the computational Grid* [2]. A well-known example of a computational Grid is the SETI@home Grid [3]. This type of Grid is primarily comprised of low-powered computers with minimal application logic awareness and minimal storage capacity.

The objective of Grid is to represent all heterogeneous computational resources, storage systems, databases and other special-purpose computing devices as a unified integrated resource. It offers resource owners to contribute resources to the Grid system, at the same time encourage the user to optimally utilize Grid resources. Resource providers can analyze the demand of its own resources and adjust the prices accordingly. The resources are accounted by users as CPU time, main memory, secondary storage, I/O time, etc.

In the global Grid environment, users submit their jobs to Grid Resource Broker, which then discovers resources, negotiates for service costs, performs resource selection, schedules jobs to resources and monitors their execution. The different prices are offered for Grid resource services, and those prices can negotiate using one of several economic models from the real world [2]. For enhanced utility delivered by resources, resource allocation is performed based on user jobs quality-of-service (QoS) requirements/constraints (e.g. deadline and budget constraints to minimize time/cost). In grid accounting system, all users would prefer to use powerful resources, which would cause some resources to be oversubscribed and others are undersubscribed. This is where computational economy and suitable service pricing strategies come into play. The price is raised for high demand resources; in other hand, the less demand resources have lower price. Resource owner have the permission to solicit an open market price in a way that achieves maximum profit and resource consumers are allowed to choose resources that meet their QoS requirements.

The Gridbus project [2] is developing technologies that enable service-oriented cluster and Grid computing.

It is driven by a distributed computational economy approach to the resource sharing, exchange, discovery and aggregation. Currently, access to resources is not economy-based. Resource owners and users co-operate as part of the same *Virtual Organization* (VO) [4], which is a collaboration of people agreed to share resources. For sharing computational services across multiple VOs (administrative domains) there is a need for accounting infrastructure that would allow unambiguous recording of user identities against resource usage. Moreover, VOs can choose to introduce their own currency for resource trading. In the context of Gridbus project we call such system the *GridBank.*

GridBank (GB) is a secure Grid-wide accounting and (micro) payment handling system. It maintains the users (consumers and providers) accounts and resource usage records in the database. It supports protocols that enable its interaction with the resource brokers of Grid Service Consumers (GSCs) and the resource traders of Grid Service Providers (GSPs).

2. INTERACTION OF COMPONENTS

The Figure 1 shows the interaction between GridBank and various components of Grid. GSPs and GSCs open account with GridBank [1]. Then, the user submits application processing requirements along with QoS requirements (e.g., deadline and budget) to the Grid Resource Broker (GRB). The GRB interacts with GSP's Grid Trading Service (GTS) or Grid Market Directory (GMD) to establish the cost of services and then selects suitable GSP. It then submits user jobs to the GSP for processing along with details of its chargeable account ID in the GridBank or GridCheque purchased from the GridBank. The GSP provides the service by executing the user job and the GSP's Grid Resource Meter measures the resources consumed while processing the user job. The GSP's charging module contacts the GridBank with request to charge the user account and transfer the funds/credits to the GSP account. It also passes information related to the reason for charging (resource usage record). The *Grid Resource Meter* (GRM) module will interface with local resource allocation system (e.g., cluster scheduler) or user-level Grid agents (such as Nimrod-G agent [5]) to extract resource usage information. The interfacing can be mediated through Grid middleware tools such as Globus, by extending it such that a native operating system usage function is called after a user application finished execution, and another function that is called by GRM to collect the data. Once GRM obtains the raw usage statistics, it chooses relevant fields in the record and passes them to the conversion unit, which generates a standard OS-independent Resource Usage Record (RUR).

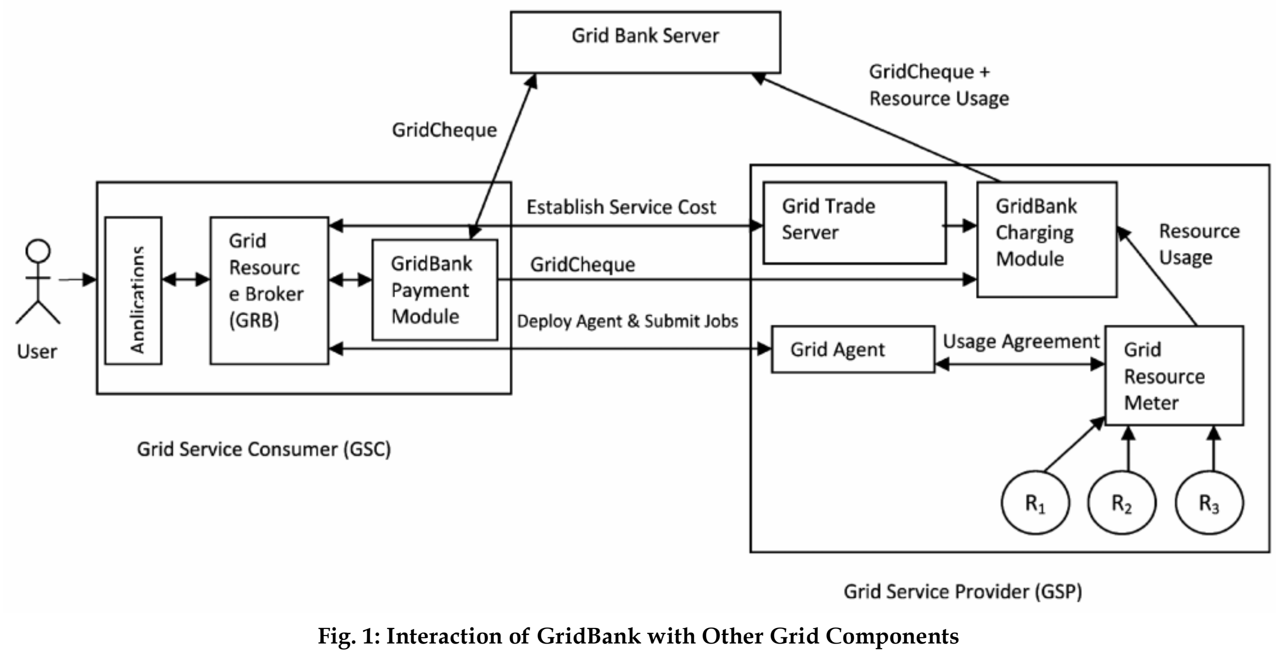

Fig. 1: Interaction of GridBank with Other Grid Components

1. GRB negotiates service cost per time unit (e.g. $ per minute)
2. GridBank Payment Module requests GridCheque for the GSP whose service GSC wants to use GridBank issues GridCheque provided GSC has sufficient funds.
3. GridBank Payment Module forwards GridCheque to GridBank Charging Module.
4. GRB deploys Grid Agent and submits jobs for execution on the resource.
5. Grid Resource Meter gathers Resource Usage Records from all resources used to provide the

service, optionally aggregates individual records into one Resource Usage Record and forwards it to GridBank Charging Module. Grid Resource Meter optionally performs usage check with Grid Agent.

6. GridBank Charging Module contacts GridBank and redeems all outstanding payments. It can do so in batches rather than after each transaction.

The RUR is a combined effort of the Grid community at the Global Grid Forum (GGF, RUR). GridBank stores this record in its database, which provides evidence that a transaction took place. RUR is then forwarded to GridBank Charging Module (GBCM) for calculation of the total service cost. GBCM obtains service rates for the user from the Grid Trade Server (GTS). The user ID (Certificate Name) passed to GBCM is contained in the GridCheque (or any other payment instrument). The service rates record generated by the GTS and the Resource Usage Record must conform to each other. For every chargeable item in the rates record there must be a corresponding item in the RUR. Chargeable items to be considered are [2]:

- Processors: User CPU time;
- Main Memory;
- Secondary Storage;
- I/O channels (such as networking);
- Software Libraries: System CPU time.

The total charge is calculated by multiplying rate by usage for each item and then adding up individual charges.

Apart from the basic functionality, depending on the kind of payment protocol GridBank Charging Module is using the GRM provides different levels of accounting information. Different protocols might require different resource usage statistics. For example, in Figure 1, each individual resource (R_1 – R_3) used to provide computational service presents its usage record to Grid Resource Meter (GRM). GRM might choose to aggregate individual records into the standard RUR to reflect the charge for the combined GSP's service.

3. GRIDBANK SYSTEM ARCHITECTURE

3.1. Payment Strategies

The Figure 2 shows GridBank architecture that controls existing technologies and manages them as separate components.

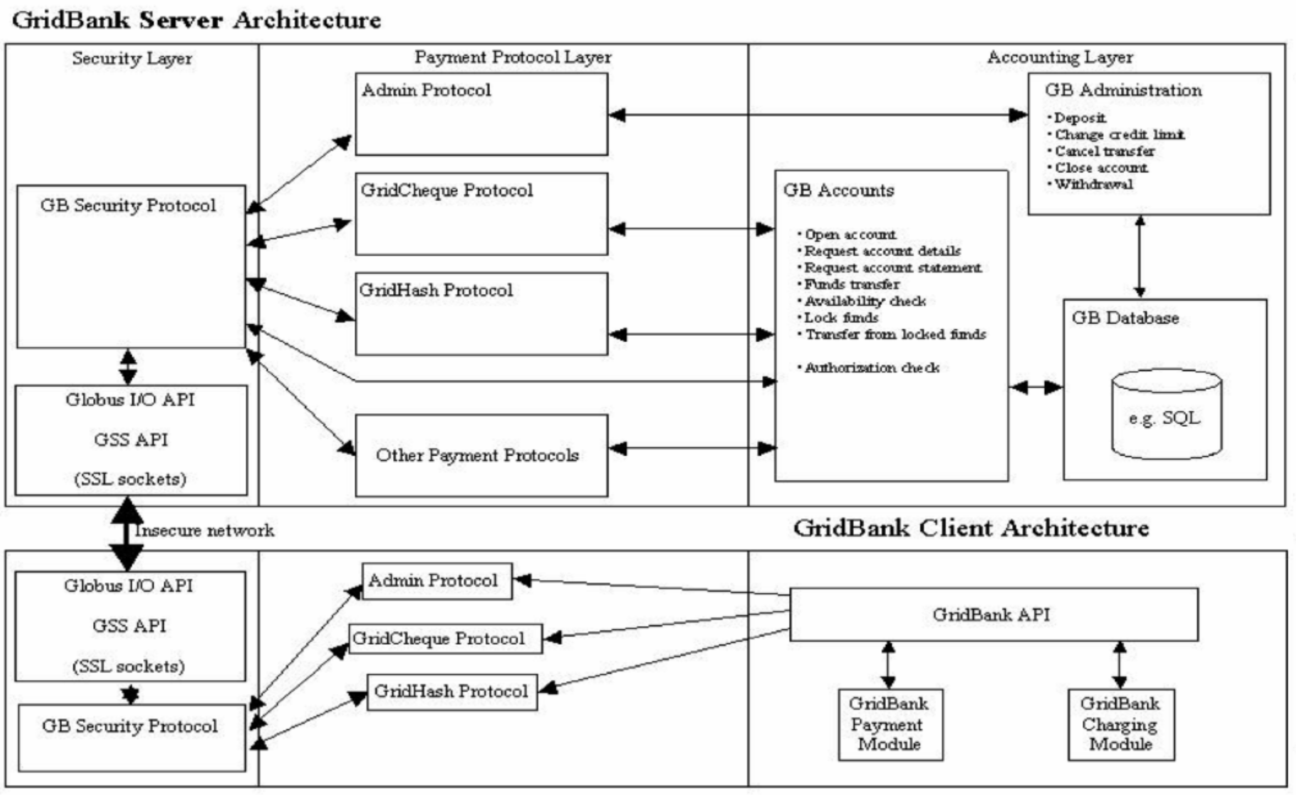

Fig. 2: GridBank Server Architecture

Depending on charging strategy, GSC and GSP can select appropriate protocol and exchange service for currency. The charging policies include [6]:

1. Pay before use;
2. Pay as you go;
3. Pay after use.

The first policy is appropriate for services that have a fixed cost, for example, to access a directory service. A simple funds transfer protocol is designed to enable GSC to request funds transfer with the confirmation send to GSP. GSC establishes secure connection with GridBank to provide account details of GSC and GSP as well as amount and URL of GSP. GridBank performs the funds transfer and sends the confirmation to the specified URL of the GSP via another secure channel.

The second policy might be used to eliminate unnecessary trust relationships between GSCs and GSPs. A hash chain scheme based on PayWord would allow service consumers to dynamically pay service providers for CPU time or per each computation result delivered [7].

The third payment strategy emulates credit card payment model. When the service charge is unknown beforehand, GSC forwards a payment order in the form of a digital cheque to GSP. The cheque is made out to GSP so no one else can redeem it. After computation has finished, GSP calculates total cost and forwards the cheque along with resource usage record to GridBank for processing. This can be done in batches. Such scheme is based on NetCheque protocol [8] and relies on public key cryptography.

When a chargeable service has a fixed price as in Pay Before Use strategy or the client obtains hash chains, there are no issues regarding availability of funds; a client could never overspend since his/her account is checked and decremented beforehand. On the other hand, when a credit card approach is taken as in Pay After Use strategy when the total cost is not known beforehand, clients can easily spend more than they have in the account (even taking into account credit limit this is also an issue if cost exceeds available balance together with credit limit). To guarantee payment when issuing GridCheques, GridBank will have to lock a certain amount of funds for the cheque to be valid. The exact amount will depend on the budget constraint set with the GRB. Each GSP will receive a cheque with a reserved amount, which is transferred to the "locked" balance of the GSC's account.

3.2. Server Architecture

Several modules that can be enhanced or replaced without affecting other modules constitute GridBank's server (Figure 2). These modules are organized into three layers. Accounts Layer deals with database and account operations. Payment Protocol Layer defines payment schemes, message formats and communication protocols. Security Layer ensures that any messages passed to Payment Protocol and consequently invoking operations in Accounts Layers are authentic.

GB database module is a relational database that stores account and transaction information.

GB Accounts is the core module interacting with the GB database. It provides functions for basic account operations such as creation of accounts, requesting and updating account details, transfer of funds from one account to another, locking funds and transfer from locked funds.

GB Admin module provides account management such as deposit, withdrawal, change credit limit, cancel transfers and close account functions. These functions are performed by GridBank's administrators who are responsible for transferring real money to and from clients.

GridCheque Protocol module implements the pay-after-use scheme.

GridHash Protocol module implements pay-as-you-go payment strategy. Pay-before-use protocol does not involve generation of any payment instruments. Transaction is performed on-line. Secure connections guarantee authenticity of participants.

GB Security Protocol module performs authentication and authorization of GridBank's clients. Authentication process uses Generic Security Services (GSS) API, which is implemented by Globus Toolkit I/O module (GLOBUS I/O API).

3.3. Client Architecture

The Security Layer is identical to the server. The Protocol Layer has same protocol modules as the server with corresponding client functionality. GridBank API provides an interface to the Protocol layer, which is responsible for obtaining payment instruments or performing direct transfers. GridBank Payment Module and GridBank Charging Module interface to GridBank API module to invoke GridBank operations.

4. IMPLEMENTATION DETAILS

4.1. Database Records

The following records constitute GridBank's database, designed using MS-Access:

Account Record:

- AccountID: Imitates real world account numbers: bank number-branch number-account number. E.g. 01-0001-00000001;
- CertificateName: Globally unique client identifier;
- OrganizationName: Optional;
- AvailableBalance: Represents Grid credits/ dollars/money units;
- LockedBalance: To guarantee payment for jobs that already have started;
- Currency: e.g. GridDollar, USD, INR;
- CreditLimit : Default is 0.

Transaction Record

- TransactionID : Unique transaction identifier;
- Type: Possible types are: Deposit, Withdrawal, and Transfer. In case of transfer, there is a corresponding transfer record with the same TransactionID;
- Date: Date when transaction was committed to database;
- Amount: Amount of transaction: if withdrawal or transfer from the account, then the amount is negative.

Transfer Record

- TransactionID: Same as in the corresponding transaction record;
- Date: Date when the transfer was committed to database;
- DrawerAccountID: GSC Account ID;
- Amount : Transfer amount: always positive;
- RecipientAccountID : GSP Account ID;
- ResourceUsageRecord : Contains RUR in a binary format.

The format of RUR (Resource Usage Record) is being defined by the Grid community effort (GGF, RUR). Currently, the following items are being associated with RUR:

- User details;
- Host name / IP address;
- Certificate Name (Grid-wide unique ID of GSC);
- Job details;
- Job ID (Local process id on the resource or GRID global unique id);
- Application name;
- Job start date (includes time);
- Job end date;
- Resource details;
- Host name / IP address;
- Certificate Name (Grid-wide unique ID of GSP);
- Host type (optional);
- Local job id (local OS process id to settle disputes about resource consumption);
- Wall clock time + price per time unit (e.g. Per second);
- CPU time + price per time unit;
- Main memory + price per time unit;
- Secondary storage + price per time unit;
- Network activity + price per time unit;
- Software service + price per time unit;
- Total price per time unit;
- Job Cost (= (end date - start date) * total price per time unit).

4.2. GridBank API

GridBank API module (Figure 2) defines the following operations for GridBank Payment Module and GridBank Charging Module; implemented using JSP and HTML.

- Create New Account

  Input: Client's Certificate (Certificate is checked for authenticity; if legitimate, then Certificate Name is extracted)

  Output: AccountID
- Request Account Details / Check Balance

  Input: AccountID

  Output: ACCOUNT RECORD
- Update Account Details

  Input: ACCOUNT RECORD (First, Request Account Details operation returns current record, a client amends it and invokes this operation to update the record. Only CertificateName and OrganizationName can be modified.)

  Output: Confirmation.
- Request Account Statement

Input: Account ID, Start Date, End Date

Output: Account Record, Transaction and Transfer Records (from Start to End Dates)

- Perform Funds Availability Check (For protocols where payee requires confirmation of a certain amount; the amount is transferred into locked balance for guarantee)

  Input: Account ID, Amount

  Output: Confirmation

- Request Direct Transfer

  Input: From AccountID, To AccountID, Amount, RecipientAddress

  Output: Confirmation sent to Recipient Address

- Request GridCheque

  Input: AccountID, Amount

  Output: GridCheque

- Redeem GridCheque

  Input: GridCheque, Resource Usage Record

  Output: Confirmation

- Request GridHash chain

  Input: AccountID, Amount

  Output: GridHash chain

- Redeem GridHash chain

  Input: GridHash chain, Resource Usage Record

  Output: Confirmation

4.2.1. GridBank Admin API

- Deposit funds (GridBank administrator receives funds via existing credit/debit/smart card payment systems, and deposits same amount into GridBank account)

  Input: AccountID, Amount

  Output: Confirmation

- Change credit limit

  Input: Credit limit amount

  Output: Confirmation

- Cancel Transfer

  Input: Drawer AccountID

  Output: Confirmation

- Withdraw (The withdrawn funds will be transferred to an actual bank account)

  Input: Drawer AccountID

  Output: Confirmation

- Close account and get outstanding balance transferred to another GridBank account or actual bank account.

5. EXPERIMENTAL SETUP AND RESULTS

To simulate the activities of Grid Banking System, we have established a simple LAN connected setup consisting of three machines connected with a switch; one is selected as a server and rests of two are selected as two clients. Two clients are treated as Grid Service Consumer (GSC) and Grid Service Provider (GSP). Using *socket* the connection is established between server and clients. Here TCP/IP protocol is used.

User should register to the Grid system and accept the all terms and condition of the system, then it can get membership of the Grid, and acts as Grid Service Provider or Consumer. Grid accounting system generates a unique user id for each member.

Functions of Grid Service Consumer (GSC):

- User provides the login id and password;
- The Server checks its database to find if the user is a member of Grid or not; if the use is member, control comes back to consumer end;
- Then GSC puts its request for task with amount and deadline (i.e. QoS) to Server; request may be deposit, withdrawal, transfer or update account operation.

Functions of Grid Bank Server:

- During whole process of job execution, the Grid Bank Server should be in active mode;
- It receives the GSC's request;
- Selects the appropriate Grid Service Provider (GSP) for required resources to process the task depending upon the QoS;
- Server finds the CPU utilization of GSP; if GSP is executing the other important task, the Server passes that request to another member of Grid;
- During the whole process, GSC does not know which GSP does this task;
- At the end of process, Server forwards the amount of CPU time usage, memory usage & cost of job done.

Functions of Grid Service Provider (GSP):

- Provider updates the database and fulfills the request (account update, deposit, withdrawal) of Consumer;
- Grid Resource Meter calculates the CPU time, memory usage, I/O usage of the whole process and calculates the amount which will be paid by Consumer and sends it to the Server.

6. CONCLUSION

We have studied and simulated Grid components in the context of the Grid-wide banking service that enables users to engage in global computational economy. We have designed a simple setup consisting of three machines connected with a switch; one is selected as a Grid Bank Server and rests of two are selected as Grid Service Consumer (GSC) and Grid Service Provider (GSP). Grid Resource Meter extracts resource usage information from the operating system and converts it into a Grid-wide standard form.

In the future, GridBank system will be expanded and tested to provide multiple servers/branches across the Grid to achieve scalability. In other words, we will have to simulate the system using multiple servers and clients across different networks.

REFERENCES

[1] Barmouta A, Buyya R (2002): "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration", *International Journal on Distributed, Parallel, and Cluster Computing*.

[2] Buyya R (2002): "Economic-based Distributed Resource Management and Scheduling for Grid Computing", PhD Thesis, Monash University, Melbourne, Australia, April 12, 2002.

[3] Sullivan W T, Werthimer D, Bowyer S, Cobb J, Gedye D, and Anderson D (1997): "A New Major SETI Project based on Project Serendip Data and 100,000 Personal Computers", *Proceedings of the Fifth International Conference on Bioastronomy*.

[4] Foster I, Kesselman C, Nick, J, Tuecke, S (2002): "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration". http://www.globus.org/research/papers.html

[5] Buyya R, Abramson D, Giddy J (2000): Nimrod/G: "An Architecture for a Resource Management and Scheduling System in a Global Computational Grid", *Proceedings of the HPC ASIA '2000, the 4th International Conference on High Performance Computing in Asia-Pacific Region*, Beijing, China, IEEE Computer Society Press, USA.

[6] Buyya R, Abramson D, Giddy J (1998): "A Case for Economy Grid Architecture for Service Oriented Grid Computing", *Proceedings of IPPS/SPDP '98 Heterogeneous Computing Workshop*, pp. 4-18.

[7] Rivest R, Shamir A (1996): "PayWord and MicroMint: Two Simple Micropayment Schemes", *CryptoBytes*, **2**, No. 1 (RSA Laboratories, Spring 1996), pg. 7-11. Also in the Proceedings of 1996 International Workshop on Security Protocols, (ed. Mark Lomas), (Springer, 1997), Lecture Notes in Computer Science No. 1189, Pages 69-87.

[8] Wayner P (1997): Digital Cash. 2nd Edition. AP Professional, Academic Press Limited. ISBN 0-12-788772-5.