BISSA - A Scalable and Distributed Tuple Space

Charith D. Wickramarachchi[1], Dulanjanie Sumanasena[1], Pradeep R. Fernando[1], Udayanga S. Wickramasinghe[1], Gihan Dias[1], Srinath Perera[2] and Sanjiva Weerawarana[2]
[1]Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka.
[2]WSO2 Inc., Sri Lanka.

Abstract— **The idea of tuple spaces is based on the white-board design pattern & made its first appearance in the late 1980s. Tuple space provides content addressed associative shared memory abstraction for the processors accessing it. Tuple spaces can be used to time and space decoupled communication between the processes.**

In our work, we have implemented a distributed and scalable tuple space middleware infrastructure called BISSA that can be used for decoupled communication between applications. The BISSA application scope span from browser based applications to java applications. This capability is given by two major implementations; a distributed hash table (DHT) based peer to peer tuple space implementation and a web browser based tuple space implementation.

In this paper we present and discuses our implementation methodology, test results and possible applications of the middleware.

Index Terms— tuples, Linda, gadgets, shared spaces, DHT

I. INTRODUCTION

After the introduction of the tuple space [1] model there have been much research done on the domain of tuple spaces. One main branch of this research is the attempt to make a scalable model of tuple spaces. On the other hand, research is going on to implement tuple spaces in different environments. MobiSpace [2] is an example research that has been done to implement tuple space in JavaME environment. Our intention through this research is to combine these two paradigms together to build a more versatile tuple space model that can span across different platforms such as browsers, standalone Java applications, services, clouds, etc. while retaining the scalable features of a tuple space in a distributed environment.

The tuple space design provides a convenient programming model for communication between applications, compared to general message passing models. In Message passing models, parties that communicate with each other have to be alive at the same time. The advantage of tuple space communication is that the parties that communicate using the tuple space need not be connected at the same time to engage in the communication. Furthermore the processes can coexist in the space/network without knowing each other's existence. These two factors sum up to making tuple space model time and space decoupled, a feature very few parallel/distributed processing architectures posses.

While conforming to these features of a generic tuple space model, we try to describe our implementation of BISSA in this paper, in two folds. One fold is a browser based tuple space implementation which provides a communication and coordination middle-ware for browser applications. It is intended to be used as a scalable solution for a black board pattern based communication between web browser applications such as for Web Gadgets operating inside a single page, gadgets located within tabs or even for browser instances located in different geographic locations. The Other fold is a peer to peer tuple space implementation that can be used as a tuple space based communication middleware for java applications.

The BISSA browser tuple space is based on JavaScript and provides a Java script based API to access the Space. The API enables applications to act as a standalone local tuple space (residing within browser memory) or to seamlessly integrate with the distributed peer to peer tuple space outside its local context. The BISSA browser space is also implemented as a shindig [3] feature so that it can be used as an inter-gadget communication infrastructure.

BISSA peer to peer space is a DHT based tuple space implementation for Java applications. It will provide a distributed shared memory abstraction for applications, hiding underlying communication complexities from the application developer. Tuples added to the system will be distributed among the nodes. Operations of the peer to peer tuples space are implemented to tolerate the ad-hoc nature of the peer to peer systems and support fault tolerance.

II. RELATED WORK

A. *The Linda Model*

In mathematics, a tuple is an ordered set of elements which may or may not be of the same type. In computing, a shared memory space depicts a logically shared memory that is physically distributed. The Linda model, developed by David Gelernter and Nicholas Carriero at Yale University describes the concept of a "tuple space", that is a shared memory space that stores tuples. It demonstrates the concept of decoupled communication between processes. According to the Linda model, if two processes need to communicate with each other they do not send messages or share a variable. Instead, they create a new object, named a tuple and place it in a common space, rather like a white board, where an interested party may come by and extract or read the message. The model describes four basic primitives that enable the users of the space to operate on the tuples or the space itself; eval, out, in and rd[4]. The "out" operation is used to add a tuple to the space and the "in" operation is

used to extract the tuple from the space. The "eval" operation is used by the initial process to generate the processes that would be using the space for its operations while the "rd" operation is used to read the tuple without removing it from the space. This initial model has been attractive mainly due to its simplicity, and due to the model's other strong features of orthogonally, and the spatial-and temporal-decoupling of concurrent processes [5]. Since its initial idea the model has undergone many improvements or additions due to extensive research, with changes made to its primary operations, such as blocking and non-blocking reads and writes being added to its primitives. The Linda model has been tried and tested in many languages including C, FORTRAN, postscript and Scheme as base languages.

B. DTuples

DTuples is a peer to peer tuple space implementation built on top of distributed hash tables and based on the Linda model [6]. It adopts a fail restore fault model to restore a crashed node's state. DTuples' persistent tuple space consists of two levels, the Common level and the Subject level [6]. Both these levels are subsets of the tuple space. The Common level is accessible by all nodes, while the subject level is accessible only to agents that are bound to it. This will by default include the agent that created the subject and any other agents that later bind to it. The lifetime of a subject depends on the lifetime of the agents bound to it and its expiration time. The expiration time is counted when all agents bound to it leave the space. If such an agent returns to the space within the expiration time it can retain the subject and the tuples held in it. The implementation uses four primitives, in, read, out and copy-collect, used to copy tuples between the spaces.

DTuples uses Freepastry as its underlying overlay network implementation and handles fault tolerance by replicating the tuples and making all primitives transactional, employing JOTM as its transaction manager. It matches its tuples using the name of the tuple as its first argument. This name is used to get the hash key. Therefore a tuple matching the same template have the same name and is stored in the same nodes [6].

C. FreePastry

Freepastry is a scalable, distributed routing overlay for wide-area peer-to-peer applications [7]. It is implemented in pure Java language and the source code is available under an open source license. The network of Freepastry nodes is called the pastry ring. Pastry assigns a 128 bits long unique node id to each pastry node in the ring. The first node to join a particular pastry ring starts or bootstrap a new ring & new nodes join the existing ring by referring to a current pastry node in their bootstrap process. Pastry is a DHT implementation. Hence, given a <MessageKey,message> pair, it routes the given message to the node which has the closest numerical id value to the message Key. Furthermore each pastry node maintains a list of nodes that are numerically closest to the particular node. This set is identified as the leaf [7] node set.

Pasty's routing algorithm finds the numerically closest node id using prefix based discovery. It can route a message with a given id to its numerically closest node in under (log2bn, ceiling) in normal operation (b is a configuration parameter with typical value 4).

There are few well known peer-to-peer applications that have been built using the pastry routing overlay. Examples [8] include PAST: a peer-to-peer archival storage, SCRIBE: group communication/event notification and SQUIRREL: co-operative web caching.

D. Hierarchical tuple spaces

One approach to a tuple space implementation is the Hierarchical Tree Structure [9]. Generally the nodes in Hierarchical tuple Spaces are categorised into two types, namely Execution nodes and the Memory Nodes. Execution nodes would be running distributed processes in them and would be connected to parent memory nodes (arranged in a tree like structure), which in turn would be contributing to the distributed shared memory where tuples are stored or replicated along.

The hierarchical tuple space employs a hybrid replication scheme[9] for availability/fault Tolerance and a data coherency protocol for concurrency control. Basic primitives supported by this kind of a tuple space implementations are OUT (writing tuples to space), READ(querying tuples from space) and IN(removing tuples from space) . Whenever a primitive is executed on a process, the respective tuple would traverse up the node hierarchy, replicating that tuple until a match (or root node) is found. If successful, the node where correct match occurred would start propagating the matching tuple down the tree to the execution node who initiated the tuple query (ie:- IN/READ).

Improved extensions to this hierarchical scheme can be seen with visibility scoped sub tree structures [9]. Here the replication is done up to a certain depth (not upto root) of the tree so that tuple space clusters are built having their own tuple space memory. As a result execution nodes will be limited to a specific portion of the whole distributed environment and nodes outside the respective visibility scope won't be able to join in to the distributed application. Although Hierarchical tuple spaces could reduce coupling between memory distribution strategy and application programming certain limitations exist, including poor scalability, static inflexibility of the deployment model and a processing overhead and complexity involved in implementing visibility scoped tuple spaces.

E. Web Gadgets

Web gadgets are html /JavaScript/CSS content that operate within a single entity inside a web based application. They are basically specified in an XML declaration conforming to a specification standard, hence enabling it to be reused among different web platforms/applications without much of a problem. Being considered as independent contexts (even if they reside in a single web page) web gadgets can be utilised in a wide variety of applications spanning from a simple widget or reusable

components to full blown web applications integrating multiple gadgets across Internet. The XML declarations of a gadget are processed by a context known as a gadget container. It is a gadget container's responsibility to render gadget layouts and controls, process meta-data, user preferences etc. and deploy 'features'[3] as specified by declarative XML syntax of a gadget.

F. Shindig

Apache Shindig[3] is an OpenSocial[10] compliant gadget Container , providing the infrastructure needed to host and access XML based gadgets. Shindig comprises of a JavaScript based gadget container which provides core gadget functionalities such as gadget communication, layout, security etc., a gadget Rendering Server which outputs XML based gadgets into JavaScript and HTML and an OpenSocial container and Data server for OpenSocial based gadget communication .

Shindig provides different capabilities to gadgets it renders, through a mechanism called "Shindig Features". Shindig features are JavaScript libraries with some useful functionality provided either to gadgets, containers or both at the deployment time. Currently supported features [3] by Shindig vary from UI functionalities and content types support (i.e.:-Flash) to supporting OpenSocial API's. Gadgets that need to use a specific feature should indicate them within their respective gadget headers.

Shindig Pubsub, a feature intended to facilitate inter gadget communication, provides a unified infrastructure through publish-subscribe messaging paradigm for XML based gadgets. This is similar to what we are going to achieve through BISSA browser tuple Space based communication. Shindig primarily supports three primitives, publish (), subscribe () and unsubscribe (). As mentioned before analogous to publishers and subscribers paradigm, "Shindig Pubsub" associates gadgets through a set of keys or channels, which would be used in publishing and subscribing to information that would be communicated between gadgets, effectively acting as an inter-gadget communication medium.

Whenever a gadget publishes information with in a specific event, all the subscribed gadgets relevant to that specific channel are notified and the event information is passed down. Gadgets can also unsubscribe at any point in time so that they are cut-off from information coming through the respective channel.

G. Tuple Space in Mobile Environments

After the tuple Space concept was introduced the implementation of tuple Spaces in different environments has been a popular area of research. Work has been done on implementing tuple Spaces on Mobile Environments. Project LIME (Linda in a Mobile Environments) [11] focuses on implementing a Linda-like tuple Space in mobile environments. It handles the complexities that comes with the resource constrains and the ad-hoc nature in the mobile environments and provides a simple interface to the programmers.

MobiSpace [2] is a middleware infrastructure that provides a distributed tuple space abstraction for Java Mobile Edition environments. Mobile programmers will be able to use this model to achieve loosely coupled communication using a standard programming model.

MobiSpace provides a primary – based replication where all the mobile nodes will be connected to a central computer where tuples are stored. And mobile nodes will be acting as secondary servers and they will be storing a selected portion of the space in the local space. The communication happens using the GPRS service, or any other Internet service available for mobile. MobiSpace also supports replication among the peers where mobile devices have formed a peer to peer network.

H. PAST

PAST is a large scale peer to peer persistent storage utility [8] that uses pastry as its routing overlay. Each stored file in PAST has a unique id attached to it. Applications can store a file within the peer-to-peer network by giving an id & the file to be stored. Upon receiving the file, PAST routes the file as a pastry message. Once the file received by the appropriate pastry/PAST node it gets saved & replicated among the pastry leaf-set of that particular node. PAST can withstand high churn due its effective replication management system. PAST also supports the caching functionality for most recently used files. Each PAST node will have a uniform number of files if the id distribution of the files are uniform, thus ensuring dynamic content balance over the peer-to-peer storage.

Further descriptions & evaluations of PAST can be found in [8][12]

III. THE BISSA PEER TO PEER SPACE

A. Approach

BISSA global space is a peer to peer distributed tuple space. It uses Freepastry[7], a distributed hash table based routing overlay to route the content based on their hash values. The global space consists of nodes connected in a peer to peer fashion and contributes to the space. Applications which use the BISSA middleware will be connected to this peer to peer network and it will get an abstraction of a local tuple space, while in reality the tuples get stored in a distributed manner within the peer to peer network. In our implementation we used PAST [8], a DHT based archival storage and extended it for our specific needs.

Tuples are stored in the peer to peer network as PAST contents. To support tuple space operation we introduce a data structure called Index. Indexes are used to locate tuples for the tuple space lookups. Index structure is also stored in the space as PAST contents. Both tuples and indexes are replicated in the network accoutring to a user defined replication factor to achieve fault tolerance.

B. Primitives Supported

1) Write Operation

BISSA offers an operation "write" to insert a tuple in to the space. Underlying actions such as distributing and replicating data will be invisible to the user.

As an overview we can describe the write operation in two phases,

1. Write tuple as a DHT element with an associated hash calculated for it.

2. Update the Index files that are distributed in the system.

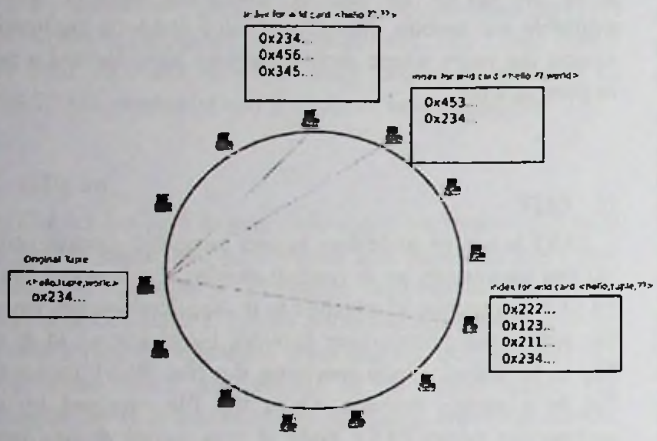

Fig 1. Wildcard indexing in BISSA

In the first phase we generate a 160-bit unique hash for the tuple and store it in the DHT. We use PAST[12], A storage system built on top of Freepastry for storing tuples. Hence the tuple is stored in a node in the system which has the node id closest to the tuple hash.

The second phase is to update the tuple Indexes. We use an indexing mechanism to support the read operation of tuples.

As shown in the above diagram after a tuple is inserted to the space we update the indexes that represents it and matches the possible wild card queries for that tuple.

Ex : a tuple : <hello,world> can be quarried from a two tuple templates <hello,??> or <??,world>

In our system a wild card is represented by a null value. So a template tuple for <hello,??> will be <hello,null>.

The tuple indexes are actually list-like data structures that contain unique hash ids for a given template and a list mapping the template ids to the hash ids of actual tuples and the nodes where they are stored.

Update of these indexes happen at the node where indexes are actually stored. In an earlier version of BISSA implementation, we used to retrieve the data structure in to the node where a tuple write takes place. The approach introduced inconsistencies & scalability issues. Thus in the current implementation we use a message passing mechanism to do the updates at remote nodes.

To improve the performance we allow parallel updates in the index storage system within a node. But to achieve mutual exclusion and thereby avoid problematic scenarios as described above we use a local-locking mechanism based on a hash id, inspired by a similar implementation in PAST. It provides the facility to access the storage system parallel but it only permits update on a same index hash id one at a time. This guarantees the fact that there can be parallel updates on different indexes in a same node but it does not allow to update the same index in parallel.

Using this implementation we have achieved the mutual exclusion property on parallel updates while not unnecessarily constraining the access.

In our current implementation we support primitive types and Java object as tuple elements. Also we provide a way to user to specify non wild card elements with the tuple insertion so that space that will be used for indexing will be saved and also the Message traffic will be less, improving the system performance.

2) Read Operation

Read operation returns the tuples that match the template given as a parameter.

Ex : <hello,world> and <hello,tuple> will be returned if user gives the template <hello,null>

(Given the fact that <hello,world> and <hello,tuple> are in the Space)

Operation of the read operation is the simplest one. In that, first the system will generate the hash for the tuple template. And then it will lookup for the index structure associated with it. So that index will have all hash ids for the real tuples matches the template. Then System will get the real tuples using that hash ids and return to the user.

In our implementation we support two versions of the read operation,

1. A blocking read where users call the read is blocked till all the tuples are received.

2. Non blocking Asynchronous read there tuples received will be given to the user at the time they received. So that users will be not waiting till all the results available. They can register a call back handler with the operation and results will be given to that handler.

3) Take Operation

This operation differs from the read operation, since it removes all the tuples that match the given template. Removing a tuple instance that has been replicated among the nodes of a peer-to-peer system raise some serious questions.

The BISSA take operation can be divided in to three phases,

1. The retrieval of the tuple from the peer-to-peer system that matches the given template.

2. Deletion of the tuple content from the peer-to-peer system.

3. Updating the indexes that carry wild card info of the deleted tuple.

4. Return the tuple to the user application that called the operation.

We use a messaging based mechanism to delete the stored tuples right in their local nodes & avoid race conditions through hash id based locking mechanism as with write operation. Still the BISSA tuple Space implementation does not guarantee the atomic execution of the take operation. It is possible to implement 100% consistent take operation

with some agreement protocol, but still we prefer light-weight behaviour of the system as our goal is to implement highly scalable tuple space implementation.

The non-atomic behaviour does not break the consistency of the space at any time. For an example if the take operation failed during the index updating phase then there will be false indexes pointing to a non-existing tuple. When read operation encounters an invalid index, it ultimately does not return the indexed tuple since it is not stored within the peer-to-peer storage.

The other major obstacle for removing all the instances of a particular tuple is the caching feature. When a node retrieves a tuple it gets stored in their local caches. The next time when there is a query for that exact tuple it retrieves the tuple from the local cache. We have provided a time stamped based caching implementation for the users who wish to make use of the cache. Users can specify the time-out of the cache, so that user don't mind getting a cached version of the tuple out from their local caches given that the threshold time has not expired (even if the tuple has been removed in the space by a take operation).

C. *Tuple Subscription Mechanism*

Bissa provides a subscription mechanism for the applications which enables them to subscribe for a tuple templates. Subscription can be done by giving a tuple template and a reference to a TupleListener implementation.

Whenever a tuple inserted in to the space, which matches a subscribed template, the subscribed parties will be get notified by the system.

There are some problems to be solved when implementing this mechanism.

1. How to distribute subscriptions in the peer to peer network.
2. How to maintain the subscriptions with the ad-hoc behaviour of the space.
3. How the un-subscribing will work

One approach is to busy wait on the subscribed tuples. But such approach will introduce a lot of unnecessary message passing in the system and degrade the performance of the overall system. As a solution for that we maintain a data structure that is associated with the tuple indexes to keep subscriptions. When a tuple added to the system which matches a subscribed tuple template; in the corresponding write operation it will update the index data structure associated with the subscribed template. After that system will send notifications to subscribers that are registered in that index structure .This subscription mechanism will reduce the number of messages passed compared to the polling method.

Maintaining the subscriptions with the ad-hoc nature of the system is another problem to be solved. With the ad-hoc nature it can cause two major problems.

1. Nodes that keep the subscription data may be unavailable or dead so that can cause inconsistencies (mainly data losses).
2. Nodes that have subscribed for data can die without properly unsubscribing. That will cause system to do unnecessary work since it will be sending unnecessary notification messages to nodes that are not there.

To avoid the problem of loosing subscription data with the dead nodes we replicate both indexes and subscription data. That will give some level of fault tolerance for subscription mechanism and other operations.

Identifying dead nodes can be achieved by running a messaging protocol time to time. But as same as the polling method it will be costly in terms of system performance. As an alternative, without polling we do a checking for liveliness of a node only when it is needed to be notified. Till then it will remain as a subscribed node, even though it may not exist in the system. If the liveliness check it returns false system will remove the subscription data for that particular node. Still yet there can be a trivial scenario that will be not identified by above method which is if a node goes dead and another joins with the same node id and in that dying and joining time period no liveliness check was done. So system will not know that it's a new node and subscription will remain. So as a solution when a node joins it sends a join message to other nodes so that nodes that has subscription data can use this detail to update its data.

IV. BISSA GADGET COMMUNICATION

A. *Approach*

BISSA browser Tuple Space resides in a local browser instance and integrates into the global Tuple Space. However depending on application programmers' choice BISSA browser space can remain as an independent tuple space residing only in browser or as a fully integrated tuple Space that is being synchronised with dynamic tuple Space environment as well. Local tuple Space API is fairly consistent with the global tuple space; hence all the primitives available in global space are true for local space as well.

Design approach of local tuple space is primarily concentrated on managing tuples that are being queried and updated on the local space and synchronising local tuples corresponding to the global space. The core functionality is implemented in a JavaScript library compatible in running under latest browsers. As is in the case of global Tuple Space, a tuple of the form {a1,a2,a3.....:an} is the granularity of message passing process in local browser based tuple Space.

Local tuple Space can be considered as a combination of a tuple pool (TP) that keeps track of tuples in local space and a hash table instance (TPHT- tuples to processes hash Table) that associates tuples with the respective local processes who involve in the tuple exchange. Whenever a tuple is written into the local space, the tuple pool is updated accordingly and relevant processes are notified of the availability of tuples using the hash table. If a local process wants to query a tuple T or a specific template Te (a tuple with wild card entries in it), Local tuple Space Manager will match T or Te with the tuples set {T1,T2,....,Tn}in TP.

A special case occurs when matching tuple set with a given template Te .First matching algorithm compares the no

of entries with each tuple in the set. If this is successful on a given tuple Ti in the set then tuple entries contained within Ti are individually matched bypassing any wild card that is encountered. If a mismatch occurs at any given entry, matching on Ti is considered unsuccessful. Ti is considered successful if an only if all the entries are matched with entries of Te.

B. Integration

BISSA provides access to global tuple Space for users who work within local browser instances .The main idea is to provide a uniform Interface that can seamlessly communicate with both local and global tuple Space memory easily and flexibly. Integration approach for BISSA infrastructure for local browser clients is through Web Services that wrap BISSA runtime instances. Prospective clients who wish to query tuples distributed in the global space (i.e.:- in different geographic localities) can use the JavaScript API provided by local tuple Space runtime inside the browser instance. This mechanism also provides an efficient way of communicating between two or more browsers located in two different machines or even within two browser windows in the same machine.

BISSA local tuple Space connects to a BISSA web service node via a JavaScript Stub that wraps an Ajax connectivity layer. The browser tuple Space will try to synchronize its localised tuple content during each connection session as required effectively appearing to be a tuple cache for local browser processes. As a result any process those queries for a certain tuple or a tuple template will be delivered all the conforming tuple content from the local tuple space as well as global Space. This model is pull based since browser tuple Space itself is responsible for pulling tuple content/data out from the globally scalable tuple space. The diagram in Fig 2 describes this overall model of integration

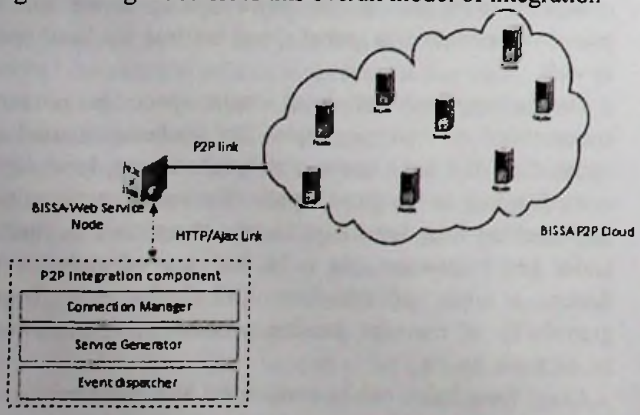

Fig 2. Overall model of integration

C. Messaging Layer

Local space queries for tuple content provided by the API are differentiated into two categories, synchronous and asynchronous. Synchronous method calls would wait until global Web service enabled global node replies for the tuple query while asynchronous calls would notify the node and resume execution. Users have the choice of adapting any of these variants depending on the requirement or performance considerations.

Sometimes it may be important that the browser application relieves the connection burden on global tuple Space and resume the local work it has been engaged in to increase performance of the application. BISSA asynchronous connectivity supports this requirement by temporary delegating tuple queries to global layer and letting it do the work for you. The local tuple content will only be updated when these queries become successful reducing overall memory footprint on the browser runtime.

On the other hand, sometimes applications may want to constantly query content n a specific template. In this context handling and management of tuple queries would be pretty cumbersome to the application developer .Application programs can use subscribe option in these scenarios, to bind into global space for specific tuple queries. BISSA will take care of the periodic querying on global space for the respective tuples/templates and notifying the relevant processes which subscribed/bind to them.

D. Web Gadget Communication

With all these aforementioned features at our disposal, BISSA has the potential to become an extremely efficient communication infrastructure for web based applications and scripting based platforms .Usually web application integration spanning across the internet tend to be ugly and highly complex and coupled with the increase of number of integration units. This inherent nature of web applications can be exploited by BISSA to provide an ideal way of inter-communication, since this is exactly and fundamentally what BISSA tries to resolve, "Providing a shared memory abstraction for naturally tightly coupled set of processes/entities". So in-effect communication between these set of application entities/components can be facilitated through writing and retrieving tuples through BISSA (shared) tuple Space, greatly reducing complexity or coupling between the components involved.

In implementation aspects this concept of using BISSA as an inter-communication framework has presented us with several challenges. That is, where/in which aspect of web we should be implementing such a system? And how exactly it should be implemented to achieve an almost unified way of inter-communication? The answer lies in Web Gadgets. As stated previously on this paper, Gadgets are web content that operate within a single entity inside a web based application. Specified in xml syntax they are considered independent entities even when they are contained within a single web page. So as per the answer to our first question web gadgets seems to be a very good choice due to their independent and highly coupled nature. For example BISSA can present a shared channel for gadgets operating inside a web page or set of gadgets integrated across a network/internet. That in-effect is an efficient communication mechanism since otherwise complex message passing mechanisms could have deteriorated the overall system performance.

Web Gadgets also seem to be the ideal answer to our second question as well. The solution lies in "features" which are a standard compliant way to implement specific services for web gadgets. This is in the sense that, BISSA browser/local space runtime can be wrapped into a gadget feature [3], to be deployed on gadget servers at runtime.

Therefore gadgets can be deployed in different gadget servers (provided we have required BISSA feature implementation for each different gadget server i.e., Shindig, iGoogle, etc) exposing them to specific BISSA runtime libraries creating a unified infrastructure for inter-gadget communication. Using this kind of BISSA enabled feature, gadget developers could use the functionality described above to facilitate communication between gadgets located in a single browser window (is:-using local space API) or even between different gadgets located in separate servers or domains.

Following is high-level overview of how a BISSA enabled feature can be deployed on a Gadget Server. Note that this feature requires BISSA local Tuple Space runtime to be present in both container and gadget level at the deployment time. This is because gadgets and their containers are typically decoupled in gadget servers and gadgets are accessing local Tuple Space instance shared between them. BISSA feature is using RPC feature intern to delegate API calls between container and gadget level.

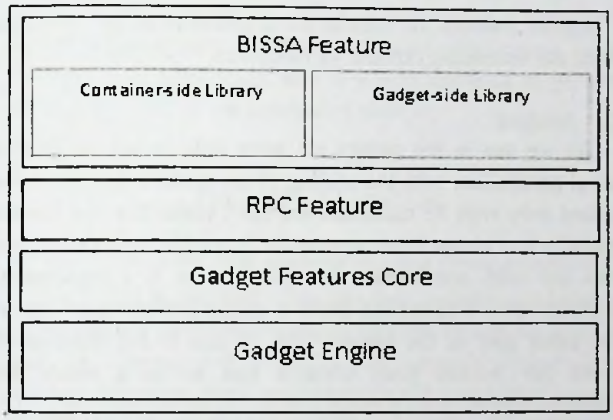

Fig 3. Overview of the BISSA features implementation.

E. The Big Picture

If we predominantly focus on the big picture, what BISSA tries to create is an infrastructure that expands over intra-nets, possibly over the Internet on different platforms where each node is contributing to the distributed memory of the tuple space

With local and global space integration in place we can see some of these nodes exposed as BISSA Web Services enabling independent browser instances to act as clients providing them the ability to access global space as well as their own local ones. This is shown in Fig 4.

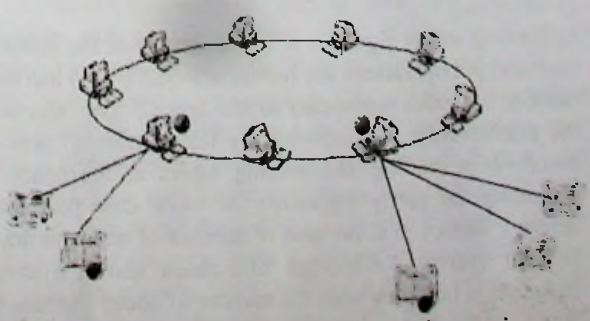

Fig 4. BISSA local and global space integration

In turn these Local tuple space instances would provide the inter-gadget communication infrastructure for Web based gadgets so that multiple gadgets participating can form very robust and powerful web based applications integrated across a geographically dispersed area. The fact that now these gadgets are effectively connected across a scalable global tuple space provides them obvious advantages such as persistence and scalability. Additionally BISSA enables different clients other than browsers to connect to Global tuple Space as well. This is also achieved through the web-service interfaces of BISSA web-service nodes connected to BISSA cloud (or ring) .Therefore any web service enabled or REST based client such as mobile devices, Java clients, data centres, etc. can use tuple Space as a unified way of communicating between different platforms or systems. In essence BISSA provides enormously wide range of possibilities to web/gadget application developers and different platform designers of various kinds for applications that span from parallel computation to distributed system integration and coordination.

F. Functions Supported

BISSA local space provides a clean and consistent API to support inter gadget communication. Although related to generic tuple space API, functions supported makes into two categories.

a) Local tuple space API

b) Global tuple Space Integrated API

This differentiation gives application developers a clear separation of concerns whether to use a standalone local tuple space or to get exposed to the distributed tuple Space.

1) Local tuple space API

Following are the supported local space primitives.

a) bissa.read(t_template,callBack,subscribe)

Read a tuple from the space relevant to the given template.

*callBack = the call back function that the requested tuple will be delivered when available in space or immediately, if already existing in space.

*subscribe = if true the user will be notified every time a tuple with the given template is added, else notified only when tuple is available in space OR immediately, if tuple is already in space.

t_template can be either a tuple template (i.e.:- bissa.Tuple("a","??","??")) or a pure tuple (i.e.:- bissa.Tuple("a","b","c"))

b)bissa.take(t_template,callbck,subscribe)

Remove the tuple from the Space and read it

callBack = the call back function that the requested tuple will be delivered when available in space or immediately, if already existing in space.

subscribe = if true the user will be notified every time a tuple with the given template is added and respective tuple

will be deleted from space else notified only when tuple is available in space OR immediately, if tuple is already in space and will be deleted from space after that .

c)bissa.put(tuple)

Insert a tuple into space; subscribed users will be notified if requested tuple is being inserted.

2) Global tuple space API

Following are the Global Space primitives supported,

a)bissa.read_global

Read tuples to a given tuple/template from the tuple space. Tuples would be read from both global and local space. This API supports both asynchronous and synchronous messaging .Synchronous read requests to the tuple space would block until an available tuple is fetched from the global space. Asynchronous reads would immediately switch control to local space after dispatching the respective request to the global space. Tuple read requests can be bind/subscribed to a template. Tuple space will take care of fetching tuples from global space from periodic intervals for a specific timeout period. User can configure these parameters to suite their application requirement.

b)bissa.take_global

Removes tuples to a given tuple/template from the tuple space .Tuples would be removed from both global and local space. Similar to the #global_read this API supports both asynchronous and synchronous messaging.

c)bissa.put_global(tuple)

Insert a tuple into both global space as well as local space.

V. RESULTS AND ANALYSIS

After successfully implementing the tuple space service we performed a scalability test and a latency test to measure the behaviour and performance of the peer to peer network while scaling up the system. The results and analysis of this test are as follows.

A. Scalability Test

The application we used for the test was a Monte Carlo simulation [13] for stock value changes for a company. This is an embarrassingly parallel algorithm, which involves independently executing disconnected components [14]. We used BISSA as a middleware to share tasks between processors and finally to aggregate the results; using a master/slave strategy where at start master will put tasks to the shared space and then after a starting command by the master, workers will take tasks from the space in parallel, execute them and submit the finished work to the space.

We used a control lab environment with machines with same configuration and did the scalability test with up to 35 computers.

1) Results Obtained

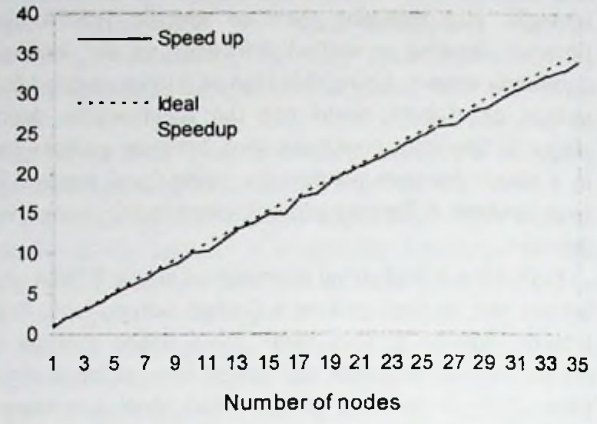

Figure 5: Speed up gained with the increasing number of nodes.

Figure 5 shows the Results we obtained from the scale test with the increasing number of machines.

2) Analysis

As we see in the results we were able to get an almost ideal parallelism with the scaling of the system. But since we tested only with 35 machines we can't claim that we have a massively scalable implementation. But with these results we can say with confidence that our system is a reasonably scalable one. The results show a slight performance lag in the latter part of the results. This is due to the number of work per worker node became less so as a result the communication latency came into effect on the results.

B. Latency Test

One other test performed on the BISSA peer to peer space was the Latency test. A latency test is used to measure time taken for an operation to complete. The idea of this test was to measure the time taken for the main BISSA operations; write, take and read to complete and analyse the effect of increasing number of nodes to its performance.

1) Results Obtained

Figure 6 shows the results obtained in measuring the time taken to perform one read and one write on an increasing number of processes.

2) Analysis

According to the figure 6 it is apparent that the latency of read and put operations are hardly affected by the increase in number of nodes connected to the space. This shows that the performance is unaffected by the number of processes involved in reading and writing tuples to the space. But according the pastry algorithm the worst case the message delivery latency is a function of number of nodes in the peer to peer system. Although this result shows a minimal variation in latencies with the number of nodes introduced,

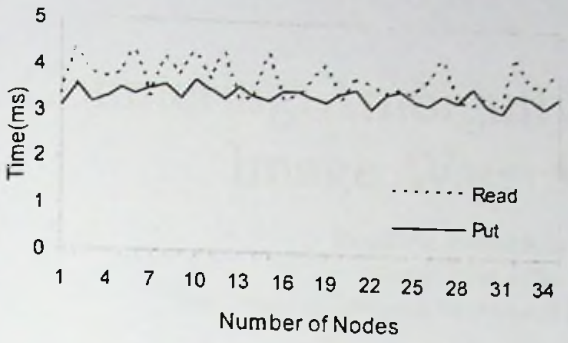

Figure 6: Operation latency with increasing number of nodes in the network.

we must expect the latency of the operations to be increased when the number of nodes is large.

According to research conducted by Kato and Kamia[15] Freepastry behaves reasonably well up to about 800 nodes. Hence we can reasonably conclude that, BISSA being built on top of Freepastry and having showed good results to the tests we have conducted, that it would continue to perform well for an even larger number of nodes.

VI. APPLICATIONS

A. *The Role of the Web Browser in Distributed Processing.*

The world is moving from standard desktop applications to web-based applications. The modern operating systems such as Chrome OS establish this concept further. With browser becomes more prominent in the computing market, we should explore the possibilities of getting the processing power of the browser in to distributed processing environments.

Such applications can make use of BISSA as their underlying communication middleware. The peer to peer global tuple space will act as the shared memory between all the browser instances. JavaScript processing clients can fetch data from the space, process them & write back to the global space.

B. *BISSA for SETI @ Home Like Applications*

SETI at home like distributed computing efforts lacks a user friendlessness to some extent since users need to download a client and contribute to the grid. But there can be considerable about of people who are willing to contribute and can't contribute since it requires them to download an application to the local computer and run it.

One solution we are purposing is to use the computation power given by the web browser. If we take modern web browsers it facilitates execution of Java scripts. System can give a URL to a user and it can go to web application that can run a Java script base client (ex: web Gadget) so that gadget will get the jobs from a job pool and calculate and send back the results. So it will be a better solution than telling people to download an application. BISSA can be use to implement this kind of infrastructure easily since it provide a unified communication mechanism for Java applications and Java script gadgets.

C. *Simple Weather App*

The Weather Monitor is yet another possible example application of possible applications of BISSA. It utilises the distribution of BISSA to gather and analyse weather data. Each node in the weather monitor may act as a weather collector, weather monitor, or both. Weather collectors add tuples with weather information along with its location to the space. For example, a weather collector collecting temperature information at location id x and tuple id y will be updating temperature t with tuples of the form <x,y,temp,t>. The weather monitor we implemented for demonstration purposes uses only temperature information, therefore its tuples take the form <x,y,t>. The weather monitors query for matching tuples in constant intervals and updates its weather information accordingly. This querying is done using individual threads that handle new information coming in from each location.

VII. FUTURE IMPROVEMENTS

A. *Extending BISSA to Mobile Spaces*

As described above in our current implementation we have a two level tuple space implementation where one is a peer to peer tuple space implementation and other is a in browser tuple space implementation which act as a client to the peer to peer space.

As a future improvement to the our model we can extend this model to have mobile nodes as explained in MobiSpace[2] So they will be acting as a client space to the peer 2 peer space. Having this kind of Mobile Space integration with BISSA will be useful since. It will introduce a unified easy to program communication infrastructure for Java Applications, Web Gadgets and Mobile Applications.

VIII. CONCLUSION

Here we presented the design & implementation details of BISSA, a middle-ware infrastructure based on tuple spaces for distributed and web based applications. We have shown how BISSA can comprehensively replace highly coupled message passing systems with a loosely coupled shared memory abstraction using an effective tuple space communication paradigm.

Here we also present how the hash based DHT addressing mechanism can be mapped to content based tuple addressing mechanism. Furthermore this paper presented how BISSA browser tuple Space can act as a local tuple cache and shared browser instance to facilitate rapid web application integration and gadget communication.

On another perspective, BISSA can be considered as a light weight distributed application platform with somewhat relaxed consistency semantics. This makes BISSA a very

powerful and scalable middle-ware for a wide variety of web and Java based applications, acting as a unified communication infrastructure to make different environments integrate easily and effectively.

BISSA is an active project with a working code base, which implements the approaches/functionalities we have discussed in this paper. The website www.bissa.sourceforge.net offers updated information about the project and downloadable resources including BISSA source code.

REFERENCES

[1] Nicholas Carriero, David Gelernter, "A computational model of everything", *Communications of the ACM*, Vol. 44, No. 11, November 2001.

[2] Fongen A., Taylor, Simon S J. "Mobispace - A Distributed Tuplespace for J2me Environments", in *the 17th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), Phoenix, AZ, USA, 2005.*

[3] The Apache Software Foundation, "Shindig- An overview of Apache Shindig", *Apache Shindig*, 03-02-2010, Available:http://shindig.apache.org/overview.html. [Accessed 25-04-2010]

[4] Nicholas Carriero, David Gelernter, "Linda in Context", *Communications of the ACM*, Volume 32 , Issue 4 April 1989, pp 444, Available: http://portal.acm.org/citation.cfm?id=63337 [Accessed 26-04-2010].

[5] George Wells, "Coordination Languages: Back to the Future with Linda", *Proceedings of WCAT'05, 2005,pp 87-98.*

[6] Yi Jiang et al, "DTuples: A Distributed Hash Table based tuple", *Proceedings of the Fifth International Conference on Grid and Cooperative Computing*, 2006, pp: 101 – 106.

[7] Antony Rowstron, Peter Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems", *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, pages 329-350, November, 2001.

[8] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", *ACM Symposium on Operating Systems Principles (SOSP'01)*, Banff, Canada, October 2001.

[9] Antonio Corradi et al, "A Scalable tuple Space Model for Structured Parallel Programming".

[10] OpenSocial and Gadgets Specification Group, OpenSocial Specification 1.0, *Opensocial, March 2010, Available:* http://opensocial-resources.googlecode.com/svn/spec/1.0/OpenSocial-Specification.xml, [Accessed 13-04-2010]

[11] Amy L. Murphy, "Lime Introduction", *LIME: Linda in a Mobile Environment, 2007, Available:* http://lime.so urce*forge.net/, [Accessed 20-04-2010]*

[12] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", *HotOS VIII*, Schoss Elmau, Germany, May 2001.

[13] N. Metropolis and S. Ulam, "The Monte Carlo method", J. Am Statistical Association, vol. 44, pp. 335-341, 1949.

[14] Geoffrey C. Fox, "Lessons from Massively Parallel Applications on Message Passing Computers", Proc. 37th IEEE International Computer Conference. NPAC, 1992.

[15] D. Kato et al., "*Evaluating DHT. Implementations in Complex Environments* by. *Network Emulator*", IPTPS 2007, 2007.