Distributed Architectures
A Comparative Analysis
Client-Server (socket), RPC/RMI, P2P, Grid
Where do you want to go today?

Chintan Odhavji Patel and Feng Charlie Zhao
University of Missouri – Kansas City, Kansas City, MO 64110

Abstract
The requirements in the business world or in scientific domain keep varying depending upon the application or task at hand, moreover in the current software scenario there are many distributed architectures available, which match the requirements. So it becomes really difficult to choose the architecture that would optimally and easily handle the task. This paper presents a comprehensive feature and performance comparison for the existing distributed architectures: Socket based Client-Server, RPC/RMI, P2P and Grid Architectures. Depending upon the empirical formulas and experimental results we try to give guidelines to make a concrete decision for choosing a particular distributed architecture under a given a set of constraints which may be computing resources, architecture complexities, performance requirements or even financial limitations.

General Terms: distributed architecture, client-server, P2P
Additional Key Words and Phrases: RMI, RPC, CORBA, DCOM, Grid, performance.

Introduction
Distributed Software engineering is becoming a de facto standard in Software development environments. The major motive behind Distributed SE lies in resource sharing and heterogeneity. There are several approaches for developing a distributed software system, which ranges from simple Client-Server to highly distributed Grid Architectures. Here is a comprehensive listing of the technologies used in distributed software engineering.

1. Client-Server Systems
2. RMI - Remote Method Invocation
3. DCE - Distributed Computing Environment -OSF
4. CORBA - Common Object Request Broker Architecture -OMG
5. DCOM - Distributed Component Object Model
6. Jini
7. JavaBeans
8. Peer to Peer (P2P)
9. Grid Architecture

Most of the distributed systems are based on client-server architecture where there is a clear demarcation between the client and the server. The client gives out requests to the server and it does the processing and returns results back to the client. Client-Server architecture has some limitations when we consider a large distributed system working on the Internet.

So to make the life easier for the developer and to decrease the development time and the production costs, the concept of middleware was introduced. Middleware is basically software that resides between the application and the operating system. It provides interoperability as well as transparent location of servers in distributed heterogeneous system [1]. Middleware is responsible for providing transparency layers that at deal with distributed system complexities
such as location of objects, heterogeneous hardware/software platforms and numerous object implementation-programming languages [3].

Before the advent of middleware based distributed software development, the complete system had to be built from scratch manually programming the connections between the systems; so most of the effort was expended in the ancillary tasks. Middleware has been introduced to provide interoperability as well as transparent location of servers in heterogeneous client-server environments.

Motivation

Today in such a Distributed world of Internet we have lots of existing technologies to harness the power of pervasiveness of the resources. Sometimes it becomes difficult to choose from the set of available technologies because all of them have their advantages and disadvantages. The DCE provides APIs where the programmer does not need to worry about the underlying details, but a major drawback of DCE is that it provides a procedure-oriented approach. Later middleware’s were developed which worked on Object Orientation model, such as CORBA, DCOM, JavaBeans etc, which are more akin to the real world scenarios. [1] Discusses the impact of client-server interaction architecture on the performance of CORBA Systems, wherein they operate CORBA architecture under different workload environments. CORBA is always criticized for performance issues but [4] makes an important point by demonstrating benchmarking of CORBA with MPI (Message Passing Interface) and PVM (Parallel Virtual Machine), finally concluding that some CORBA implementations (ORBacus and TAO) could also be used for distributed and Grid Computing applications. In our research we would like to do such an analysis, which covers such concepts related to the effectiveness of Architecture. We would also like to consider some factors such as Scalability and fault-tolerance issues related to a Architecture, [5] mentions how the current distributed paradigms, like CORBA or RPC only provide the basic protocols and services for interoperability and fail to define more complex relationships among the application’s distributed elements. Our research will focus on similar lines, which will take in to consideration the current available Architectures and come up with the weaknesses and the strength of different approaches. [2] is about fusing Metadata WWW specification of RDF in a P2P architecture. It is interesting as it makes the searching (discovery protocols) much sophisticated thereby making P2P architecture more efficient. Apart from these heavy solutions like CORBA and EJB, some new light weight approaches have been developed such as Peer to Peer Systems where the control is distributed and the entities could share resources and exchange services directly.
Fig 1. When many architectural styles fit your requirements (Where do you want to go today?)

**Problem Statement**

Our Research outcome is to give comprehensive comparisons between the existing distributed architecture [Client-Server, RPC, P2P, Grid Architecture] approaches. To give guidelines to make a concrete decision for choosing an architecture given a set of constraints which may be resources, processing, memory, or complexities (deployment or development).

**Solution**

Our solution is divided into two parts, which when combined would help us to solve the problem.

**Part A. The Experimental Approach:** In this approach we do actual implementation of the particular architecture and tabulate the results depending upon some selected criteria’s. Here we will also consider the Time and Space complexity of the algorithms used to test the architecture.

**Part B. Empirical Analysis:** Some features could not be “measured” experimentally, like the scalability of the architecture, the reliability or complexity. So to take into account these features, we will develop some empirical formulae to handle these features.

In end combine results from Part A and B and have a result showing the prominence of a architecture in a given scenario.
A. The Experimental Approach

Assumptions made:
- Service Providers (Server) contains the algorithms to process the job demanded.
- Jobs are such that they could be broken down into small jobs which could be executed separately using the same algorithm and the results which we get from all jobs could be also appended to get a complete solution.
- The jobs are generated randomly and are being computed sequentially.
- Server is never down, it always exists to process the request.

The “SCI FI” Calculator
Calculator would be a highly complex calculator that could do many more tasks than just simple calculator functions. The calculator will have two threads:

1. THREAD_SEND: This thread sends out job requests using some random number generation algorithm to generate the type of jobs randomly. We will maintain the following structure on the entity sending the requests.

<table>
<thead>
<tr>
<th>JOB_ID</th>
<th>STATUS</th>
<th>SENT TIME</th>
<th>TOTAL TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>2156</td>
<td>SENT</td>
<td>18:16:14.50, Fri 04/04/2002</td>
<td>1200 secs</td>
</tr>
</tbody>
</table>

2. THREAD_RECV: This thread receives the job results and gets the JOB-ID searches the table entry changes the status and calculates the total time required to solve the problem.

The Structure of Job is as follows:

Structure JOB
{
    Integer JOB_ID
    Integer Thread_ID
    String Machine_ID
    Vector PARAMETERS
    Vector RESULTS
    Integer TYPE_OF_OPERATION
}

- JOB_ID: This element stores a unique ID for the job sent.
- Thread_ID: This element stores the ID of the thread on the machine that generated the request.
- Machine_ID: It is basically the IP address of the machine that sends the calculation job requests.
- PARAMETERS: It is a vector that stores the parameters required to carry out the calculation job.
- RESULTS: It contains the results of the calculation performed.
- **TYPE_OF_OPERATION**: It is a unique ID that we will maintain across ALL the participating entities, which will help us to identify the type of operation to be performed on the parameters.
  - Simple calculation: Operation like addition, subtractions etc fall into this category.
  - **Discrete Inverse Fast Fourier Transform Algorithm IFFT** [14], this is the formula for calculating the FFT for given Frequency domain values to convert them in to corresponding Time domain values.

\[ f_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{-2\pi i k n / N} \]

Where,
Hn : is a vector containing discrete frequency domain values.
hk: contains the resultant vector of the corresponding time domain values.
N : its is the number of input values

This algorithm is generally used for real time calculation for frequency domain values or corresponding time domain values. These algorithms are generally implemented in the hardware. So the time complexity is a major constraint for this algorithm. The current time complexity for the DIFFT algorithm is O (N log N).
The code in Java for this algorithm is listed in Appendix A

- **Edge Detection** [15]: This is an image processing algorithm, requiring extensive memory as well as computing resources. This Algorithm is basically used in Computer Vision field where it is used to detect edges. An edge is normally defined as an abrupt change in color intensity. This algorithm requires as input a large Image as well as it is quite a bit computational Intensive. So this will help us to check the Software Architectures robustness against these factors. Code for Sobel Edge Detection Algorithm could be found in Appendix B.

We will use the same Job Structure on following architectures
1. Client-Server- This will be a pure socket based system, where in we will consider scenarios where we have a single server and multiple clients generating the requests.
2. RMI - This approach hides the underlying connection making and data serializing tasks from user.
3. P2P – We will build our own application on the platform of JXTA

Finally we will come up with a result sets obtained using different architectures. The criteria for comparisons will include following factors

1. **Calculation time**: This fundamental criterion measures the speediness of the architecture i.e. depending upon the calculation jobs load how fast do we get the results.
2. **Complexity of the tasks assigned**: We could have calculator jobs, which are simple requiring simple calculation and some intensive job such as computing using FFT algorithm. Another view of complexity could be the amount of memory required by the job, if it is some of image processing it would require large amount of storage resources.
3. **Resource usage**: This criterion measures how effective is the architecture in utilizing the available resources.

4. **Complexity of the Architecture deployment**: The distributed software architectures range from simple socket based client-server to very complex grid architectures, hence we need to consider this factor while tabulating the results.
   4.1. **Transparency**: This factor is totally a negation to the amount of complexity involved.

5. **Number of entities participating in the problem solving**: This factor takes in to account how many servers or peers we involved in getting the result.

6. **Scalability issues**: How effective is the architecture when scaled to large number of entities.

7. **Prioritization**: We will tag some calculation jobs as more important or critical than others and see how the underlying architecture supports the prioritizing the jobs, i.e. does it gives the results faster for high priority jobs.

8. **Dependence on Network QoS**: How much is the architecture depended on the services provided by the underlying network.

9. **Modifiability**: What special facilities are provided by the architecture for modifying the existing system.

10. **Fault-Tolerance**: How efficient is the architecture in providing a fault-tolerant environment for some time critical application.

**B. Empirical Analysis**

Consider following parameters:

- **Scalability** – S
- **Fault-Tolerance (Reliability)** – R
- **Complexity** – C
- **Resource Utilization** – U
- **Execution Speed** – E

\[ R \propto S \]

[Reliability is directly proportional to Scalability]

\[ i.e. \ R = KS \]

Where K is a Constant, which determines the **degree of reliability** of the Architecture when we scale it.

\[ E \propto U \]

[Execution speed is directly proportional to Resource Utilization]

\[ i.e. \ E = XU \]

Where X is a constant, which determines the **EFFICIENCY of the Architecture** considering the resources utilized and the execution time.
Open Issues

For the fair comparison between the client-server and the P2P architectures, we certainly need to address couple open issues here.

1. **Search**: There are many variations for the advertisement and query model for search in the P2P architectures [8], therefore, we also need to provide a similar mechanism in Client-Server architecture to search for the suitable working-server in the network.

2. **Job Dispatching**: If we consider a scenario for simple Socket based Client Server and RPC/RMI, it becomes an important issue for us to dispatch the jobs among multiple servers in the network. Whereas in P2P we have proprietary protocols handling these issues.

   **Possible Solution**: Middleware is all about integration. [10] We can provide a thin Middleware in Socket-based and RPC/RMI architectures to dispatch the job based on the Round Robin fashion to the available working-server. In this way to roughly equalize the capability to utilize the three valuable and fundamental Internet assets: information, bandwidth and computing resources. [7]

3. **Common Framework**: With the design of the computing environment as a dynamic grid in its heterogeneity, we are seeking the common framework, which provides the equal visibility to the client-server and P2P. And this framework is required for our comparison of the scalable, robust, efficient and dynamic distributed architecture.

Related Work

Much of work is being done in comparing the different implementation within the same architecture but very little work has been done on comparison between the Architectures. [12] discusses just the comparison between how RMI over IIOP is better that just RMI and later gives the scenarios for choosing the implementation, it doesn’t address the issue if some other architecture is more suitable leaving RMI all together. Some work [4] go about comparing the different implementation, where they discuss how some implementation of CORBA give very high performance and which later they benchmark. Some papers go about discussing only the performance of the implementation of the architecture [6,7,8] where they discuss issues in JXTA platform whereas instead those problems could also be solved by using an entirely new distributed architecture style. [13] is a really nice example which considers the whole architecture style for solving problems where they discuss how to use JXTA for building Grid Application in which they also raise many issues related to the Architecture style itself.

Conclusion

This research starts with a blank board and the results will be strictly based on the criteria’s which we have outlined. We also hope to refine the criteria or may be add some new as the research progresses. Depending on the scenario we can also assign weight to criteria’s for e.g. some factor like the calculation time will be given more weight to produce the final result. Finally depending upon the quantitative results obtained from the experiment and the empirical analysis we hope to give a fair comparison of these architectures based upon different scenarios.
APPENDIX A.
Inverse FFT Algorithm

Code for the Inverse FFT Algorithm in Java
[Source: http://www.nauticom.net/www/jdtaft/JavaiFFT.htm]
double[][] ifft_1d(  double[][] array )
{
    double u_r,u_i, w_r,w_i, t_r,t_i;
    int ln, nv2, k, l, le, le1, j, ip, i, n;

    n = array.length;
    ln = (int)( Math.log( (double)n )/Math.log(2) + 0.5 );
    nv2 = n / 2;
    j = 1;
    for (i = 1; i < n; i++ )
    {
        if (i < j)
        {
            t_r = array[i - 1][0];
            t_i = array[i - 1][1];
            array[i - 1][0] = array[j - 1][0];
            array[i - 1][1] = array[j - 1][1];
            array[j - 1][0] = t_r;
            array[j - 1][1] = t_i;
        }
        k = nv2;
        while (k < j)
        {
            j = j - k;
            k = k / 2;
        }
        j = j + k;
    }

    for (l = 1; l <= ln; l++) /* loops thru stages */
    {
        le = (int)(Math.exp( (double)l * Math.log(2) ) + 0.5 );
        le1 = le / 2;
        u_r = 1.0;
        u_i = 0.0;
        w_r =  Math.cos( Math.PI / (double)le1 );
        w_i =  Math.sin( Math.PI / (double)le1 );
        for (j = 1; j <= le1; j++) /* loops thru 1/2 twiddle values per stage */
        {
            for (i = j; i <= n; i += le) /* loops thru points per 1/2 twiddle */
            {
                ip = i + le1;
                t_r = array[ip - 1][0] * u_r - u_i * array[ip - 1][1];
                t_i = array[ip - 1][1] * u_r + u_i * array[ip - 1][0];
                array[ip - 1][0] = array[i - 1][0] - t_r;
            }
        }
    }
array[ip - 1][1] = array[i - 1][1] - t_i;

array[i - 1][0] = array[i - 1][0] + t_r;
array[i - 1][1] = array[i - 1][1] + t_i;
}
t_r = u_r * w_r - w_i * u_i;
(u_i = w_r * u_i + w_i * u_r;
(u_r = t_r;

} /* end of ifft_1d */
**APPENDIX B**

**Sobel Edge Detection Algorithm**

[Source: http://www.visc.vt.edu/armstrong/ee2984/sobel.html]

```c
#define ROWS 512
#define COLS 512

main()

unsigned char image_in[ROWS][COLS];
unsigned char image_out[ROWS][COLS];
int r, c; /* row and column array counters */
int pixel; /* temporary value of pixel */

/* initialize output image array */

/* (this is not necessary with most compilers) */
for (r=0; r<ROWS; r++)
for (c=0; c<COLS; c++)
image_out[r][c] = 0;

/* filter the image and store result in output array */

for (r=1; r<ROWS-1; r++)
for (c=1; c<COLS-1; c++)
{
  /* Apply Sobel operator. */
  pixel = image_in[r-1][c+1] - image_in[r-1][c-1]
  + 2*image_in[r][c+1] - 2*image_in[r][c-1]
  + image_in[r+1][c+1] - image_in[r+1][c-1];

  /* Normalize and take absolute value */
  pixel = abs(pixel/4);
}
```

University of Missouri – Kansas City
/* Check magnitude */

if (pixel > 255)

fprintf (stderr, "pixel value is greater than 255\n");

/* Store in output array */

image_out[r][c] = (unsigned char) pixel;

}
References:

[1] Performance of CORBA-based client-server architectures  
Abdul-Fatah, I.; Majumdar, S.  
Page(s): 111 – 127

Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, Tore Risch  
Proceedings of the eleventh international conference on World Wide Web May 2002

[3] Component technologies: Java beans, COM, CORBA, RMI, EJB and the CORBA component model  
Emmerich, W.; Kaveh, N.  
Page(s): 691 – 692

[4] Evaluating the performance of CORBA for distributed and grid computing applications  
Es-sqalli, T.; Fleury, E.; Guyard, J.; Bhiri, S.  
Page(s): 288 – 294

[5] Java 2 distributed object models performance analysis, comparison and optimization  
Juric, M.B.; Rozman, I.; Stevens, A.P.; Hericko, M.; Nash, S.  
Page(s): 239 -246

[6] Secure peer-to-peer networking: the JXTA example  
Yeager, W.; Williams, J.  
IT Professional, Volume: 4 Issue: 2, March-April 2002  
Page(s): 53 – 57

Li Gong  
IEEE Internet Computing, Volume: 5 Issue: 3, May-June 2001  
Page(s): 88 -95

[8] Search in JXTA and other distributed networks  
Botros, S.; Waterhouse, S.  
Page(s): 30 -35

Kant, K.; Iyer, R.; Tewari, V.  
Page(s): 368 -375

[10] Where is middleware  
Vinoski, S.
[11] Improving data access in P2P systems
Aberer, K.; Punceva, M.; Hauswirth, M.; Schmidt, R.
IEEE Internet Computing, Volume: 6 Issue: 1, Jan.-Feb. 2002
Page(s): 58–67

George Koutsogiannakis and J. Morris Chang
IEEE IT Pro, May-June 2002
Page(s): 41-47

[13] Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment
Jerome Verbeke, Neelakanth Nadgir, Greg Ruetsch, Ilya Sharapov

[http://www.ulib.org/webRoot/Books/Numerical_Recipes/bookcpdf.html]