2003 IEEE International Conference on CLUSTER COMPUTING
HONG KONG

QoS-Aware Adaptive Resource Management in Distributed Multimedia System Using Server Clusters

Offered by Mohammad Riaz Moghal and Mohammad Saleem Mian
University of Engineering and Technology, Lahore-54890, Pakistan

Introduction

A distributed multimedia video-conferencing application with a distributed architecture is presented in Fig 1.

In which clients are participating through creating concurrent sessions (chains of tasks). These sessions may have certain QoS-levels such as a *desired* or a highest-QoS-level, an *average* or medium-QoS-level, and a *required* or a lowest-QoS-level.

In Fig 1 three different set of lines, each set representing three media: audio, image and video and each media is having three quality levels.

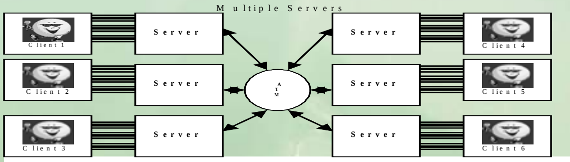

Fig 1 ADAPTIVE DISTRIBUTED MULTIMEDIA SYSTEM
Legend: Top-line = QoS_Level_1 = Desired QoS,
Center-line = QoS_Level_2 = Average QoS,
Bottom-line = QoS_Level_3 = Required QoS

ADMS System for Cluster Computing

The Background and Service Model

In an Adaptive Distributed Multimedia System (ADMS) shown in Fig1, there are n concurrent sessions (task chains) created by clients that share m resources.

If sufficient available resources are available then the ADMS should provide each session the desired-level of QoS. Otherwise, it must negotiate the operating QoS of a set of sessions.

The adaptation of QoS in an ADMS may be initiated by many different events, such as start of a session, drop of an existing session, change of system load, change of network load and dynamic changes of user's choices/requirements.

The main difficulty in an ADMS is to work out the operating quality *qi* of each session *i* which maximizes the system service *S* by minimizing the BRU under given system resource constraints.

This minimizing BRU approach improves both: 1) acceptance ratio, 2) system response.

User QoS Profile.

When a user creates a session *i*, she has a quality profile which is expressed as Q*i* = (q1, q2, q3) where q1 is the highest desired QoS and q3 is the lowest required QoS the user is willing to ccept shown in Table1.

Mapping User-QoS to Resource-QoS.

We map from an operating quality to the resources required by the session, the resources are represented by *R (qi)* where *R (q)* denotes the quality to resource mapping (*f:* Q → R) function, see the Table 1 for mapping.

Session Acceptance and System Serviceability.

The system serviceability is equal to sum of all the individual sessions' utilizations. The service function-*S* of a system maps a session's operating quality ***qi*** to session service *s(**qi**).* The system serviceability function *S* known as: $S = \sum s_i (q_i)$.

QoS Resource Bottlenecks.

The sum of the quantities of the resource allocated to all the sessions for a given Quality qi must not exceed the total available quantity of the resource. If RT is the available system resources then there source constraints are defined as: $\sum R(q_i) \leq RT$.

TABLE I
ADAPTABLE USER'S PREFERENCES FOR ADMS FOR DISTRIBUTED MULTIMEDIA APPLICATION

Media	Desired*(q_1)* (Highest-Level) = $ 60	Average *(q_2)* (Medium-Level) = $ 30	Required*(q_3)* (Lowest-Level) = $ 10
Audio	Surround	Stereo	Mono
Image	High Resolution	Medium Resolution	Lowest Resolution
Video	High Resolution	Medium Resolution	Lowest Resolution

Quality q1 may be mapped to 60% of CPU cycles, 60MB of Memory, 60 Kbps network bandwidth;
Quality q2 may be mapped to 30% of CPU cycles, 30MB of Memory, 30 Kbps network bandwidth;
Quality q3 may be mapped to 10% of CPU cycles, 10MB of Memory, 10 Kbps network bandwidth

Service Model

The main features of the Service Model (SM) are as follows:

1) The user states quality profile,
2) The session's operating qualities are then mapped onto the required resource
3) A session's operating qualities are mapped onto the session's utilities.
4) The system service is the sum of all session utilities.
5) The system's allocation decision is subject to resource constraints.

The main difficulty in the system is to work out the operating quality *qi* of each session *i* which maximizes the system service *S* by minimizing the BRU under given system resource constraint. This minimizing BRU approach improves both the sessions acceptance ratio, and 2) system response.

The SM can be used for:

1) Session Acceptance: *S'>S, where, S'* = system Service with n+1 sessions and *S* =
2) system Service with n sessions, 2) QoS Adaptation, 3) Integrated Resource
3) Management, and 4) Priority Based Resource Allocations:

Problem Formulation and Our Heuristic Solutions

We need to partition each session (chain of tasks) into two sets, one for the attached client to the chain and the other for server, as shown in Fig 2. A cut-in-chains correspond to an allocation which is subject to multiple constraints is an NP-complete problem. Our algorithm finds all the multiple-sum paths in a layered graph of chains for such problem.

The load is dynamically shared among clients and server (s) in such a fashion that minimum BRU is encountered. The SQ algorithm was designed for the single-host-satellite system. The host is like a server and satellite is a client in a client–server system. Then the same problem is solved using another double-server-client distributed architecture using our RED algorithm.

We present a new multiple-server-client (cluster-server client) architecture along with a new algorithm REMS for multimedia video-conferencing application

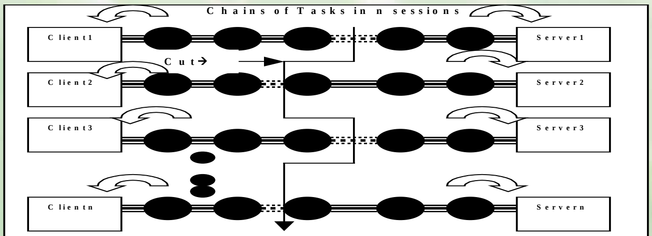

Fig. 2. Dynamic task chains cut/partitioning and mapping onto Clients and Server machines. Left-subset-side of the partitioned task chain is mapped onto the attached Client and the Right–subset-side onto the Server.

Our REMS Algorithm:

```
Input: c = # of chains of tasks or # of sessions;
     m = # of cuts in each chain;  each cut with a 2k-dimensional load-vectors Left-side-of-chain-cut for  client-machine and Right-side-of-chain-cut for Server machine.
     k = # of constraints;  L = normalization value (Between 0 to 1);  ClientId= Cp, where C = # of Clients; 1≤p≤c
     n Servers = S1-Server1; S2-Server2,… Sn-Server n
Output: Set of Cuts { }; Bottleneck resource usage BRU;  Bottleneck resource id BR = Best Cut
Step (A): The Client and Servers load-initialization
                              for all Cp, set used load-vector to 0
                              for S1 or S2,…,Sn set used load-vector to 0
Step (B): Calculate initial cuts for chains
for (i=1; i <= c; i++){
  for (j=1; j <= m; j++){
    vector-add load Vector of chain i at cut j to load of  Cp and S1   or S2, … Sn
    get the current BRU; if current BRU lower than best BRU so far; store j as best cut so far and update best BRU
    remove load of chain i from Cp and S1 or S2 or…Sn
  }  vector-add load of chain i at the best cut found to load of Cp and S1 or S2 … Sn
}
Step (C):  improve iteratively the selected cuts
for( iteration=1; iteration=m-1; iteration ++){
  for (i=1; i <= c;  i++) {
    remove load of chain i from Cp and S1orS2 or… Sn as implied by best cut so far find best cut & worst cuts from
    chain i as in Step (A)
}
    mark worst cut as disabled;  vector-load of chain I at the best cut found to load  of  C and S1 or S2 …Sn.
}
Step (D): compare result
for ( i.=1; i≤ c; i.++)
C=best-cut-for-Best-Load-Balance-after-last-iteration-of-Step(C);
BRU = bottleneck resource usage after last iteration of step(C)
```

Experimental Results and Simulation

The results show that in terms of minimum relative BRU, our REMS algorithm with three-Servers outperforms with 9-11.31% than RED and 30.3-31.3% than SQ algorithms with different problem sizes. And for four-servers REMS outperforms with 26-28.80% than RED and 43.93-44.45% than SQ algorithm.

TABLE 2
Relative BRU with the following problems sizes for servers cluster computing

# of Server	Algorithm	Relative BRU(%)	Approx.Time Complexity(sec)
1	SQ	29.14	35
2	RED	22.55	36
3	REMS	20.31	37
4	REMS	16.28	38

# of Sessions = 4 # of Tasks in a Session or in a Chain = 4
# of Cuts in a Chain = 4 Random Capacity of a Client Machine = 800 To 2400
Random Capacity of a Server Machine = 3200 To 9600, Random Load of each Task of a Chain = 100 To 300

TABLE 3
Relative BRUu with the following problems sizes for servers cluster computing

# of Server	Algorithm	Relative BRU(%)	Approx.Time Complexity(sec)
1	SQ	58.29	35
2	RED	45.00	36
3	REMS	40.44	37
4	REMS	32.90	38

# of Sessions = 4 # of Tasks in a Session or in a Chain= 4 # of Cuts in a Chain 4 Random Capacity of a Client Machine = 800 To 2400 Random Capacity of a Server Machine = 3200 To 9600 Random Load of each Task of a Chain = 200 To 600

TABLE 4
Relative BRUu with the following problems sizes for servers cluster computing

# of Server	Algorithm	Relative BRU(%)	Approx.Time Complexity(sec)
1	SQ	87.52	35
2	RED	67.80	36
3	REMS	60.13	37
4	REMS	48.61	38

# of Sessions = 4 # of Tasks in a Session or in a Chain = 4 # of Cuts in a Chain = 4 Random Capacity of a Client Machine = 800 To 2400 Random Capacity of a Server Machine = 3200 To 9600 Random Load of each Task of a Chain = 300 To 900