MQSeries

Design considerations for large Clusters

Version 1.0

11th October 2000

John B Jones
PISC Software Solutions
Hursley Park
Winchester
Hampshire
SO21 2JN

johnj@uk.ibm.com

Take Note!

Before using this report be sure to read the general information under "Notices".

First Edition, October 2000

This edition applies to Version 1.0 of *MQSeries - Design considerations for large Clusters* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

Trademarks and service marks

Acknowledgments

The following individuals contributed in varied ways to developing the presented information.

- Mick Lickman, my partner in developing this approach
- Andy Banks and Anthony O’Dowd, for their help in ensuring the validity of the approach and it’s technical accuracy
- Stephen Todd, for the original idea
- David Hardcastle, Sarah Greasley and a customer with whom they were involved, for giving us the opportunity to develop this approach with a real scenario
- Steve Eggleton and Pete Symonds for the details of the real scenario
- Jeff Hooper, for his editorial assistance

Summary of Amendments

Date	**Changes**
11 October 2000	Initial release

Preface

This SupportPac is intended to address some of the challenges encountered in the design of MQSeries queue manager cluster solutions for large-scale MQSeries installations. It is based on work done with a major UK customer.

For the purposes of this SupportPac 'large-scale' is defined as follows: Any installation where, if a single cluster were used and all queue managers in the cluster needed to connect to a queue manager, the number of channels which that queue manager is required to run would be greater than the number which can be active simultaneously on that queue manager. Potential growth and expansion should be considered when deciding whether an installation is large-scale.

Some knowledge of MQSeries Queue Manager Clusters is assumed.

My objective is to supplement the reference material - not to replace it. If you need detail you cannot find here, see the formal documentation, particularly the excellent MQSeries Queue Manager Clusters manual.

Bibliography

- *MQSeries Queue Manager Clusters, IBM Corporation,* SC34-5349

Introduction

This SupportPac is intended to address some of the challenges encountered in the design of MQSeries queue manager cluster solutions for large-scale MQSeries installations. It is based on work done with a major UK customer.

For the purposes of this SupportPac 'large-scale' is defined as follows: Any installation where, if a single cluster were used and all queue managers in the cluster needed to connect to a queue manager, the number of channels which that queue manager is required to run would be greater than the number which can be active simultaneously on that queue manager. Potential growth and expansion should be considered when deciding whether an installation is large-scale.

Overview

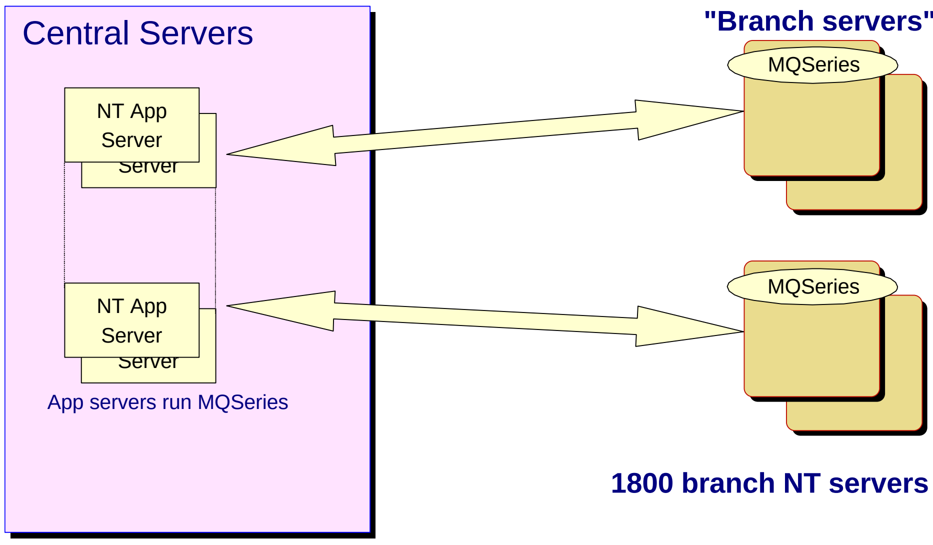
Discussed in this SupportPac are three main topics:

- Whether to use one large 'sub-divided' cluster or several clusters, each taking a sub-set of the queue managers in the cluster.
    - A single cluster with segmentation via alias queue definitions is recommended
- Whether to use parallel processing or to implement a fail-over strategy
    - A fail-over strategy using a cluster workload exit and either alias queue attributes or two overlaid clusters is recommended where the message rates from a particular client queue manager are low (measured in seconds per message rather than messages per second)
- How to manage the installation operations to ensure that the repository queue managers do not have too many channels active at the same time.
    - Plan the rollout to avoid all branches using the application for the first time at the same moment
    - Use Max Channels to handle exceptional conditions

Terminology

client queue manager	A cluster queue manager which, in the application being considered, does not host a cluster queue but uses the queues on a server queue manager.
cluster queue manager	A queue manager which is a member of a cluster. A queue manager may be a member of more than one cluster. A cluster queue manager has one **or more** of the following logical roles: client queue manager, server queue manager, repository queue manager.
server queue manager	A cluster queue manager which, in the application being considered, hosts one or more cluster queues.
repository queue manager	A queue manager that hosts a full repository of information about a cluster.

Statement of ‘Large Cluster’ problem



In this scenario, each of the central servers runs an application that the branch servers need to access. With 1800 branches accessing each server and receiving replies this would need 1800+ channel pairs active on each central server – not currently possible.

Any solution without MQSeries Clustering would require maintenance of many channels and either real-time maintenance of channels to ensure availability of the business application or that the application is aware of the available destinations.

Though the problem is expressed using Windows NT queue managers as an example the same considerations apply on all platforms, albeit at different numbers of channels.

The numbers referred to are the specific numbers involved with the customer account at which this problem, and its’ potential solutions, were investigated.

Central server machines

- Run MQSeries queue managers
  - Work requests arrive as MQSeries messages originating in a branch
  - Replies returned in an MQSeries message
- Do the application work
  - No details known but assumed to take significant machine resources and probably involving database accesses

Branches

- 1800 branches with 1 channel pair per branch gives 1800 channel pairs
  - Too many for a single Central Server MQSeries for Windows NT queue manager
  - The transaction rate from each branch is also important.
    - If each branch produces one transaction every 5-10 seconds

        - A single channel will be lightly loaded
        - Multiple channels would only have occasional use

- Need to 'segment' the work so that not all branches deal with all servers
    - That is, need each branch to work with only a subset of the servers

The need to minimise the overall MQSeries administration, together with the need for a ‘failsoft’ capability, led to an approach using MQSeries Queue Manager Clusters.

It was also very important to avoid any routine administration at the branch servers as these were outside the direct control of the administrative group and making any change in up to 1800 physically spread locations is not desirable.

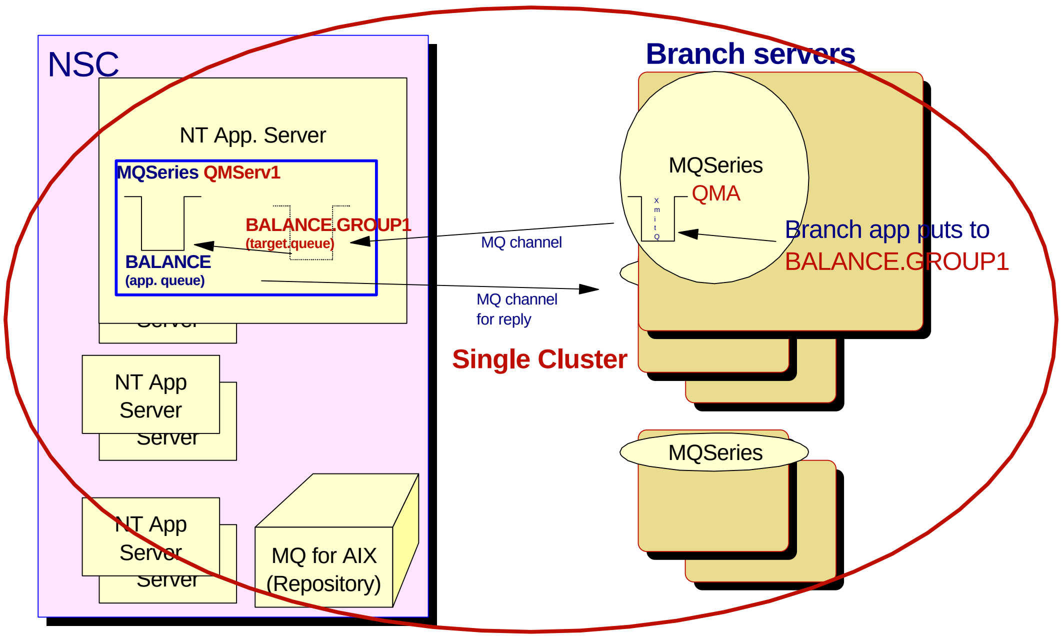
Possible Solutions

Using a single cluster

If a single cluster is used, another mechanism is needed to ensure that branches do not communicate with all the servers. The branches have to be grouped such that only an acceptable number communicate with each server.

In a situation where all branches can communicate with each server; the server queue would be made a cluster queue, and so all branches would see all instances of this queue. Where grouping is needed, the server queue is not declared as a cluster queue; instead alias queues, which resolve to the server queue, are defined as cluster queues on each server and used to group the branches. At the branch, the application puts to a destination that resolves to the alias; several ways of doing this exist and are described below. In summary:

- Achieve grouping by using a number of cluster alias queues
    - o Which resolve to the real, duplicated, application queues.



Strategies for routing to the group

- Routing by application knowing, or deriving, the group name
    - o Branches put to <application>.<group>
        - In the picture BALANCE.GROUP1
    - o Clustered alias (BALANCE.GROUP1) exists on the target server
        - Resolves to the server queue (BALANCE)
    - o Application has some awareness of its group name

- Routing by alias on Branch server
    - Application puts to the final destination queue (BALANCE)
    - Local alias (BALANCE) exists which translates this to the group queue (BALANCE.GROUP1)
    - Clustered alias (BALANCE.GROUP1) exists on the target server
        - Resolves to the server queue (BALANCE)
    - New alias must be deployed with each new application

In both cases when a new application is deployed to use different server queues, a new set of group aliases is required.

In the first case the application must pick up the group name from the environment

- Deployed with the queue manager
    - In registry??
    - In configuration file (with IP address and QM name)??
- Same technique can be used for each application
    - No new environment to be deployed with a new application

How many groups…..?

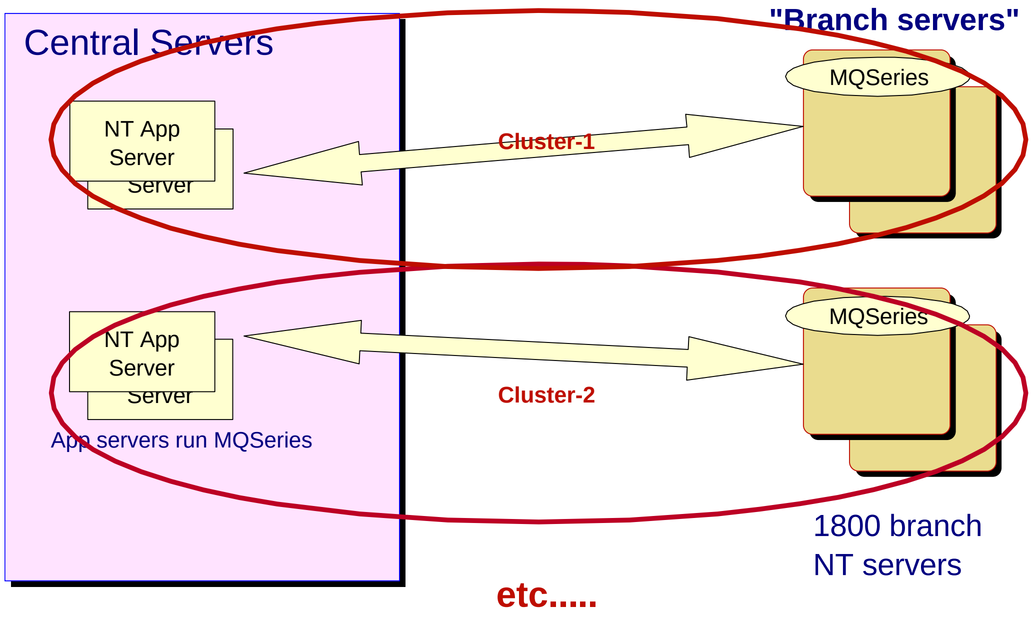
- Logically all branches could be in a single group
    - BUT, by definition of large clusters, the MQSeries queue manager being considered cannot handle that many channels
    - On the central servers, each machine would connect to ALL branches
- Logically each branch could use a unique alias name
    - BUT 1800 alias names would need to be managed across the central servers
    - Maximum flexibility but considerable maintenance and possibility of error
- Number of groups
    - There are a minimum number of machines required to allow the degree of segregation required. In the practical case we studied in most depth, it was not clear whether meeting this requirement caused extra servers to be required or whether the server application workload would have needed at least as many machines as were required to support the segmentation approach.
    - In our case we had 1800 branches and decided that 6 servers was the minimum number so at least 6 groups were needed
        - But more for flexibility

    - Not too many too maintain
    - SAY 24 -30
  - At least partially based on logical organisational groups
    - Areas ??
  - Group name for branch part of the branch set-up ??
    - But read and used by application

In the case being used as an example, the customer decided to have one cluster alias queue per branch – recognising the workload but feeling that the additional flexibility was worth the administration.

Using multiple clusters

The alternative method of grouping considered was to put the servers in different clusters, each with its associated group of branches.



Consider the administration required to balance the server workload by movement of a branch (or some branches) from a heavily loaded server to a more lightly loaded one:-

- Single cluster approach
  - Add the queue alias on the new server
  - Remove the queue alias from the old server
  - **No** action required at branch level
- Multiple cluster approach
  - Change the cluster to which the branch queue manager belongs
  - Define a channel to the repository for the new cluster
  - **All** actions required at branch level

  Adding a new server also becomes a more difficult task, as administration is required at many branch servers.

The increased level of administration at the branch level, and consequent lack of flexibility, led us to reject this approach.

Refining the design

This section describes the assumption needed to build the detailed MQSeries cluster design and includes the figures specific to the customer situation we were working with. In a general case the numbers would change but the items to be considered would not; with different numbers we may have different results.

- Assumptions
    - Fail-over capability desirable
        - Shared workload was considered but in this case was not chosen, see below for discussion
        - To minimise the amount of ‘real-time administration’ required
    - Messages were less than 30kBytes
    - Design the solution based on 300 channel pairs per WindowsNT queue manager
        - Limit 800 channel pairs from performance advice at that time (1Q00)
        - If one machine fails over to another gives 600 pairs - reasonable margin
        - 300 normally active is better because less central server machine resources are devoted to MQSeries than if 600 channels are normally active (which they would be in the shared workload case)
    - The design must be documented by the customer – to ensure understanding and buy-in.
    - Design must be sufficiently scalable to accommodate:-
        - Increased MQSeries queue manager concurrent channel capability
            - If the next MQ release can support twice as many channels, the design should be able to take advantage of this
        - Deployment of additional applications
            - Using the same or additional central servers
        - Need for additional servers
            - If the application workload estimate was significantly wrong and more servers are required
        - Need for load balancing
            - To load the central servers equally
- Discussion
    - Two aspects to consider
        - Application workload
        - Grouping of branches to even out workload
    - Application workload

    - This will be the primary factor governing how many machines will be required in the central server location(s)
        - Though a minimum number will be required to support the segmenting approach
    - Should be apparent from testing how many will be required to handle the expected branch generated workload
  - Branch grouping
    - Manual process
        - Initial estimate of traffic per branch (modified by branch activity??)
        - Assign groups to servers
    - Refine based on machine performance monitoring
        - By moving groups (reassigning alias's)
  - If a major expansion in the number of server machines is needed
    - By adding new machines in parallel with the busiest machine, rather than by rebalancing groups.
  - Deploying new applications
    - By adding new application and queues to existing machines
        - Add new alias's on the same machines
        - Possibly clone these for workload
    - By adding a new group of server machines
        - Add the new machines to the cluster
        - Add new alias's on the new machines, using the same '6-based' rules
- Result:
  - Minimum of 6 central server machines for an application
    - Scales to more machines easily
  - Adding new applications on existing server is ok.
    - As long as account is taken of grouping (not difficult if group is read from environment by common application code)
  - Adding new applications on new servers is ok.
    - As long as account is taken of grouping (not difficult if group is read from environment by common application code)

  - At least 6 servers for the new application
    Fail-over vs. balancing

At the heart of this discussion is the requirement to have no single point of failure in the installation; that is, it should never be the case where failure of a single server would cause branches to lose the ability to run a particular application.

Two ways of doing this were considered:

1. **Balancing:** Have at least two servers serving a branch equally, in case of failure all work is automatically routed to the remaining server.

2. **Fail-over:** Have one server serving a branch, and another to which the work will be routed automatically in case of failure.

Both of these approaches are possible using MQSeries Queue Manager Clusters.

In both scenarios MQSeries messages can be trapped in the failing machine; use of the term fail-over does not imply use of any of the XRF-type mechanisms, such as HACMP, for failure recovery. In either case, routine maintenance can be carried out without trapping messages by following a simple operational procedure:

1. Suspend queue manager from Cluster

2. Wait until all messages processed and replies sent

3. Stop queue manager

4. Perform maintenance

5. Start queue manager

6. Rejoin cluster

Fail-over is slightly more complex to achieve but uses significantly less server machine resources in normal operation because less channels will be active.

- If the application workload requires many more groups than the segmenting approach (greater than x2)
  - o Parallel processing will probably be used
- Else
  - o Use fail-over

If in our case the number of application servers required to process the workload had been greater than 12, we would have recommended shared workload (6-pairs of servers) instead of fail-over.

The additional factors in the fail-over case are:

- Additional MQSeries objects to administer
- A cluster workload exit is required

The function of the workload exit needed to allow automatic fail-over should be as follows:

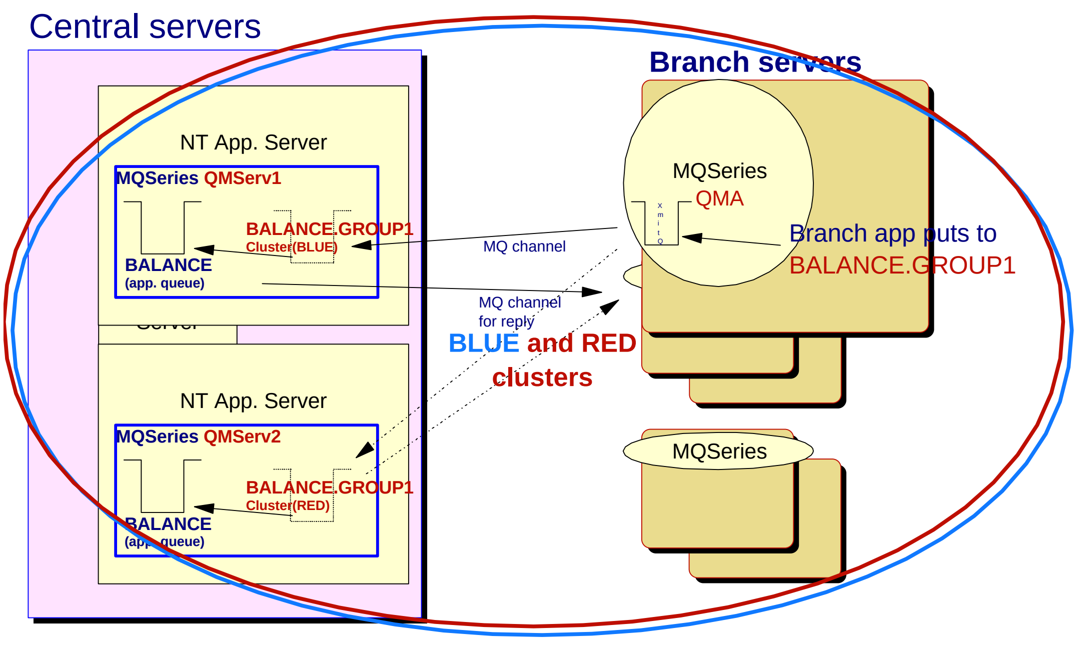
```
If the exit is being called for a PUT, PUT1, or OPEN:

        Check all valid destinations

        Accept the ‘primary’ one

             If there is more than one at this priority

                           Chose one using a random number
                           algorithm

   Else do what MQSeries suggests.
```

Coding the exit in this way keeps the logic simple, thus reducing the chance of needing to modify the exit – which will run in each branch server. If the algorithm does not cover rare exception cases specifically, the penalty is only to start an extra channel.

The ‘primary’ destination can be determined in several ways

1. Multiple overlaid clusters with different channel NETPRTY (Main and Back-up)
2. Use of DESCR (description) attribute of the cluster alias queue
3. Use of the DEFPRTY attribute of the cluster alias queue

Implementation suggestion using two overlaid clusters



- Main (BLUE) cluster is high priority (say 6)
    - o Blue - for good (fair weather)
- Back-up (RED) Cluster is lower priority (say 3)
    - o Red - for danger (we're using our backup)
- Application Server queue managers have one CLUSRCVR per cluster
    - o CLUSRCVR (BLUE) has NETPRTY(6)
    - o CLUSRCVR(RED) has NETPRTY(3) - and a different name, ending in .BU ??
- Application Server queue managers declare cluster alias's in one cluster
    - o Primary server for group declares ALIAS in cluster BLUE
        - ▪ in example QMServ1 hosts cluster alias queue BALANCE.GROUP1 in cluster BLUE
    - o Secondary server for group declares ALIAS in cluster RED
        - ▪ for example QMServ2 hosts cluster alias queue BALANCE.GROUP1 in cluster RED
- Whilst QMServ1 is available all messages put in Branch servers to BALANCE.GROUP1 are sent to QMServ1

- If QMServ1 is unavailable:- messages put in Branch servers to BALANCE.GROUP1 are sent to QMServ2

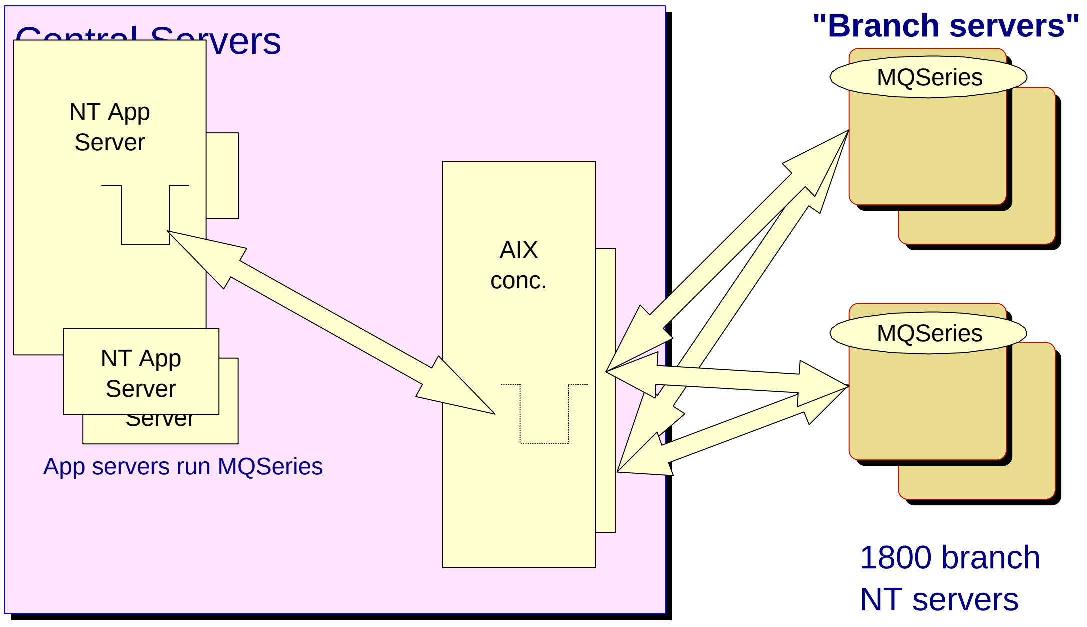
- When QMServ1 becomes available again, it starts being sent messages for queue BALANCE.GROUP1 again

One advantage of this approach is that the back-up channels can have a shorter disconnect interval than the primary ones.

Repositories

- Full Repositories
  - In production there will be little traffic on these, and it will be rare for all channels to be up together
  - In fact, I can't think of a circumstance when this need be the case
    - Given care in timing of admin activities
    - Max channels should be specified to ensure no overload on the repository
      - To minimise impact of exceptional conditions
  - CLUSRCVR disconnect interval to be tuned
  - Rollout planning to be planned to ensure that all branches do not ‘go-live’ with the application for the first time at the same time.
    - Ensure new queues exercised during a phased application rollout rather than at start of business
- Partial repositories are generated and maintained automatically by the clustering function

Alternatives Considered



- Concentrator machines
  - Using OS/390 or AIX machines
  - To handle the 3600+ concurrent channels
    - This is near the limit for AIX machines
- No problem getting messages from branch to central server concentrated
  - Or then getting the work balanced between application servers
  - Cluster alias queue on Concentrator is used as target for branch apps.
    - Resolves to the real cluster application queue on the central servers
    - Workload balanced !!
- Reply messages must be sent via the concentrators
  - Or each central application server will have 1800 + channels
    - 1800 is too many!!
- Can be achieved by queue manager alias's on the application servers
  - To route the message back to the concentrator
  - 1800 of them (but at least they don't change !!)
- Two concentrators needed or single point of failure
  - Two high specification machines required solely for routing is not a good choice where other solutions exist.

--   End of Document --