Distributed Systems

Project Name - Distributed Shared Memory

Submitted By – Aakash Singh (2021201087), Sourabh Patidar (2021201089), Karan Negi (2021201039)

1. **Brief description and algorithm used** –

   Distributed shared memory has individual nodes with individual memory modules. The programs which require access to the distributed shared memory perceive it as a single entity. The algorithm implemented here is read replication which allows data pages to be copied to the requesting node for reading purposes. This prevents unnecessary page transfers until the page is re-written.

   On reading the program tries to cache the page if it has an empty page or a cached page as the least recently used. Otherwise, it swaps. On write the program send invalidate requests for all the cache pages and then overwrite the original page.

2. Page Design –

The page is designed keeping Json in mind. It contains three entries -

a. Empty – indicates whether page contains data or not.
b. Cache - -1 indicates that the page is an original page, however any other number would indicate that the current page is a cached copy of that page.
c. Data – data here can be anything as the whole structure is encompassed in Json. Which means we can store numbers as given in sorting example and even text as used in group chat example.

3. Other data structures used -

a. Ip – contains the ip address to which the memory manager is bound.
b. Port – contains the port number to which the memory manager is bound.
c. Pages – a local dictionary of pages which contain all the page data.
d. Page_addresses – a dictionary containing a key value pair of the page number and the corresponding node id where page is present.
e. Lru – a list having arranged least recently used local pages.
f. Id – to identify the memory manager.

g. Node_addresses – dictionary containing the address I.e., Ip and port for every other node corresponding with their id.
h. Total_nodes – the total nodes currently present in the system
i. Total_pages – the total pages combined in all the nodes.
j. My_pages – the total number of pages that are present locally.
k. Cache copies – an array for every page indicating the pages which are its cache copies.

4. Basic architecture -

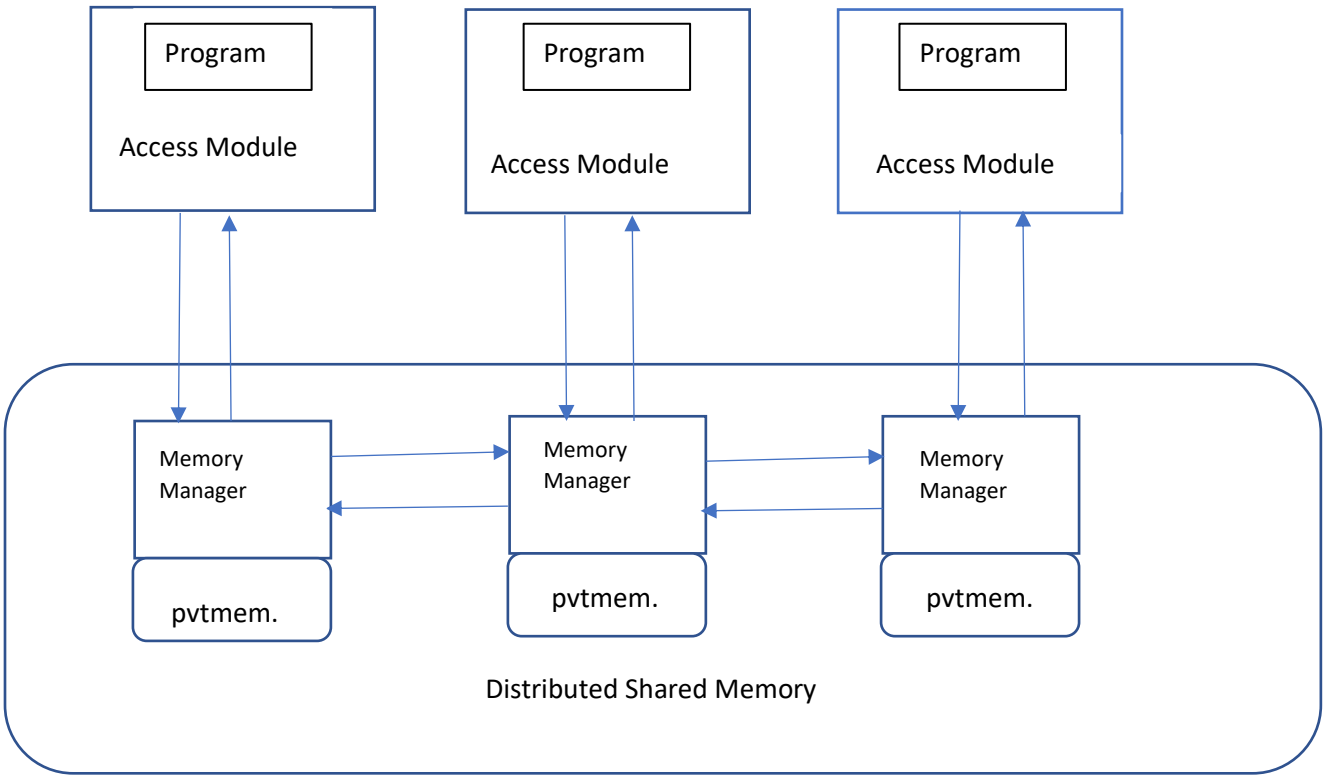
a. Memory Manager – has a defined set of pages which can be interchanged with other memory managers. They form the middle layer and provide the abstraction required to make it look to the program as if there is a single memory module. The memory manager is on a constant listening mode to receive requests from other memory managers and programs via access module.
b. Access module – a library which gives the functionality of reading and writing pages to the program which imports it. It further provides some basic info and refresh options. It abstracts socket based communication with the memory manager for the programs use.

5. Types of request messages -

a. **Asking for Neighbour config** – sends a request to copy all the neighbour details of the above data structures then adds its own pages to the mix when a reply is received.
b. **Advertise configuration** – tell every neighbour that a new node has been initiated. Asks them to update its ip, port and update the total number of pages.
c. **Read page** – generated by the access module it asks for a particular page number from the memory manager.
d. **Write page** – generated from the access module. It contains the page number and new data. It asks the memory manager to overwrite this data on the given page number.
e. **Refresh** – generated from the access module to force print the current config of all the data structures.
f. **Get details** – generated from access module to get some data structure information from the memory manager.
g. **Swap request** – memory manager asks a neighboring memory manager to swap a given page with the target page.

h. **Copy Request** - memory manager asks a neighboring memory manager to send data of a given page which It copies in a blank or cached page. This request creates a cache copy.
i. **Update page location** – a request sent after a page swap for all neighbors to update the location of swapped pages.
j. **Add cache info** – a request sent after copy request to tell all neighbors about the new cache copy of the page.
k. **Delete cache info** – sent when a cache copy of a particular page is overwritten. Asks the neighbors to update the cache copies data structure.
l. **Invalidate cache pages** – sent from memory manager during write request to all memory managers who hold cache copies of the page. It is a request to invalidate the page and treat it as empty as original data is changed.

6. Architecture Diagram



7. Initialization:

a. **Standalone:**
    i. The manager stores its Ip and port.
    ii. It initializes all the data structures mentioned above.

b. **Multiple:**
    i. Initializes its blank pages.
    ii. Connects with a single Neighbour to obtain all the collective details.
    iii. Advertises its status and pages to everyone.

8. Receiving read request from program:

a. Works based on read replication caching mechanism.
b. If the page is not with any node the manager returns sanctity error.
c. If the original page is with the node just updates the LRU (Least Recently Used) and returns data.
d. The same above pattern is repeated if we have a cached page.
e. If the page is with another manager:
    i. Send a copy request if the least recently used page is empty and make a cache copy with us.
    ii. Update cache data if the least recently used page is a cache copy. Advertise the cache change to all Neighbours.
    iii. If not swap our least recently used page with the target page of another manager. Again, advertise the new page location to all neighbours.

9. Receiving request to write a page

a. Checks for a valid page number
b. Checks if the page is with the current manager.
    i. Checks if it is empty – then just write the data.
    ii. If the page has a cached copy in it overwrite data and update every Neighbour.
    iii. If the page has cached copies send invalidation message so that the respective managers delete their cached copies.
    iv. Update the page content.
c. If the page is with another manager -
    i. Swap the page with the least recently used page.
    ii. Update all neighbors about the swap.
    iii. Perform all the steps in the above point.