An Active Replication Scheme for Mobile Data Management

Shiow-yang Wu       Yu-tse Chang

Presented by:
Jing Zhao
Yili Zhang
Litao Wan

A replication scheme determines
• the number (replication level)
• and location (replication placement)
of replicas in distributed system.
Traditional replication schemes:
Assumptions: fixed hosts, relatively static access patterns.

Static: the number and placement of replicas are predetermined and fixed.

In mobile environment:
Above assumptions no longer hold.

Dynamic: continuously maintaining statistics about access patterns and system workload so as to dynamically recalculate access cost and reconfigure the replication structure to adjust to the changes in access patterns. However, it’s still “passive” since the actual reconfiguration can only happen after the changes in access patterns have taken place for a period of time.

The active replication scheme is unique in 4 respects:

1). Maintain detail access statistics of each individual user in his profile.

2). Allow user to provide his daily schedule, so the system can provide a certain degree of predictive replication.

3). Open objects: represent a user’s current and near future data requirement. This leads to a more precise and reflective cost model.

4). Emergency events and objects: unconditionally replicated.
The access cost of most dynamic replication schemes are calculated based on accumulated read/write statistics and the chosen consistency control protocol.

Case: Deciding whether to replicate an object O on a site.

R: average num of local read requests per time unit to O on this site.

W: average number of write requests to O made by all users in the entire system.

\(\alpha\): the cost that can be saved if the object were read locally instead of requested form a remote site.

\(\beta\): the additional cost for maintaining a replica of O.

Based on this cost model, if

\[ \alpha \times R \geq \beta \times W, \]

then the allocation of a replica of O on this site is judicious.

---

<table>
<thead>
<tr>
<th>Time</th>
<th>Location</th>
<th>Activity &amp; Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:00 - 01:50</td>
<td>OR1</td>
<td>Operation32</td>
</tr>
<tr>
<td>10:30 - 12:00</td>
<td>PD1</td>
<td>Outpatient: ER3906*</td>
</tr>
<tr>
<td>14:00 - 16:00</td>
<td>MD3</td>
<td>Outpatient: ER3906*</td>
</tr>
<tr>
<td>16:10 - 19:00</td>
<td>OR5</td>
<td>Operation36</td>
</tr>
<tr>
<td>21:00 - 21:50</td>
<td>PL1</td>
<td>Project24: PR2409</td>
</tr>
</tbody>
</table>

P.S.  
MD: Medical Dep.  DR: Delivery Room  
PD: Pediatrics Dep.  OR: Operation Room  
PL: Pathology  ER: Epidemic Report  
PR: Pediatric Report  * : Emergency Object

Figure 1. An example schedule.
A dynamic replication scheme must determine

1) A consistent control protocol.
   • A cost model for estimating the access and replica maintenance cost.
   • The replication structure.
   • The replication control algorithms.

They do not place any limit on the level or placement of object replicas.

System Model

Use the **Primary Copy** model and **ROWA** (read-one-write-all) protocol for replica consistency control.

Among O-scheme (object O and all its replicas), select one as P(O);

The site that maintains P(O) must have all information (number, location) about the replicas of O; All sites that keep a replica of O must know where P(O) resides;

All reads to O can be satisfied by the nearest copy of O; All writes to O must be sent to P(O) first and then propagate to all sites in O-scheme.

The **cost of replica update** is proportional to the **size of O-scheme**.
System Information

1. **User Profiles:**
Declaring schedules and maintain per-user access statistics.

**Open Objects:**
The set of objects accessed since entering the current cell, and for
the user on schedule, the set of objects declared in the current
schedule.

Represent current and near future data access range.

2. **Replication Server**
For each user: User profile, on/off schedule status.
For each data object: Local_open_read, global_open_write.

**Local_open_read:** sum of all read histories of local users who
having this object opened. Representing the read pattern of the
current cell toward the object. Users who no longer have this
object opened are not counted.

**Global_open_write:** sum of all write histories of all users who
have this object opened. Reflecting the current write pattern to
object O in the entire system.
3. **Emergency events and objects:**

An emergency event is any situation that demands quick response. Whenever an emergency event occurs, the system unconditionally replicate all emergency objects associated with that event.

A server maintains a `local_emergency_counter` for each object declared as emergent, to count the number of emergency events or claimed users to that object. The object is replicated whenever the counter is greater than zero.

4. **System Events**

To activate replication control algorithms and the trigger the execution of the replication control code.

- READ
- WRITE
- UPDATE
- ENTER
- EXIT
- TIME-CHECK
- EMERGENCY ENTER
- EMERGENCY EXIT
Cost model

Access cost: based on network transmission cost. (read/write)

Cost of remote access: $d$, the network distance between the current site and the nearest site with a replica of the desired object.

The cost per access that can be saved from allocating a replica in the current site is $d$.

The extra cost of maintaining the replica on each update is also $d$.

The total cost saved per time unit by allocating a replica of $O$ in the current site = $d^*$ local open read;

The extra replica maintenance cost = $d^*$ global open write

If $d^*$ local open read $\geq d^*$ global open write

it is beneficial to have a replica of $O$ on current site.

- $O_1, \ldots, O_k$ are the data objects in the system.
- $B_1, \ldots, B_n$ are the base stations in the system.
- $C_1, \ldots, C_n$ are the corresponding cells.
- $U_1, \ldots, U_m$ are the mobile users in the system.
- $P_1, \ldots, P_m$ are the user profiles.
- $S_1, \ldots, S_m$ are the user schedules.
- local open read$_{ij}$ is the local open read record of $B_k$ on object $O_j$.
- global open write$_{ij}$ is the global open write record of the object $O_j$ maintained in $P(O_j)$.
- local emergency counter$_{ij}$ is the local emergency counter of $B_k$ on object $O_j$.
- read$_{ij}$ is the read histories of $U_i$ on object $O_j$.
- write$_{ij}$ is the write histories of $U_i$ on object $O_j$. 

READ Algorithm

When (the event $U_i$ reads $O_j$ occurs at $B_k$)
{
    If ($O_j$ is not an open object of $U_i$)
    {
        Add $O_j$ to the list of open objects of $U_i$;
        $local.open.read_{kj} += read_{ij}$;
        $global.open.write_j += write_{ij}$;
    }
    $read_{ij} += 1$;
    $local.open.read_{kj} += 1$;
    If (no local replica of $O_j$ exist) and
    ($local.open.read_{kj} \geq global.open.write_j$)
    Allocate a replica of $O_j$ and inform $P(O_j)$;
}

Figure 2. The READ Algorithm.

WRITE Algorithm

When (the event $U_i$ writes $O_j$ occurs at $B_k$)
{
    If ($O_j$ is not an open object of $U_i$)
    {
        Add $O_j$ to the list of open objects of $U_i$;
        $local.open.read_{kj} += read_{ij}$;
        $global.open.write_j += write_{ij}$;
    }
    $write_{ij} += 1$;
    $global.open.write_j += 1$;
}

Figure 3. The WRITE Algorithm.
**UPDATE Algorithm**

When (an update to a replica of $O_j$ occurs at $E_k$)
{
    If $(\text{local\_emergency\_counter}_{kj} = 0)$ and $(\text{local\_open\_read}_{kj} < \text{global\_open\_write}_{j})$
        Discard the replica of $O_j$ and inform $P(O_j)$;
}

Figure 4. The UPDATE Algorithm.

**ENTER Algorithm**

When (the event that $U_i$ enters $C_k$ occurs)
{
    If ($U_i$ is on schedule)
    {
        For all $O_j$ required in $S_i$
        {
            Add $O_j$ to the list of open objects of $U_i$;
            $\text{local\_open\_read}_{kj} += \text{read}_{ij}$;
            $\text{global\_open\_write}_{j} += \text{write}_{ij}$;

            If ($O_j$ is an emergency object)
            $\text{local\_emergency\_counter}_{kj} += 1$;

            If (no local replica of $O_j$ exist) and
            ((local\_open\_read_{kj} $\geq$ global\_open\_write_{j}) or
            (local\_emergency\_counter_{kj} $>$ 0))
                Allocate a replica of $O_j$ and inform $P(O_j)$;
        }
    }
}

Figure 5. The ENTER Algorithm.
EXIT Algorithm

When (the event that \( U_i \) exits \( C_k \) occurs)
{
    For all open objects \( O_j \) of \( U_i \)
    {
        Remove \( O_j \) from the open objects of \( U_i \);
        local_open_read_{kj} := read_{ij};
        global_open_write_{j} := write_{ij};

        If (\( O_j \) is an emergency object requested)
            local_emergency_counter_{kj} := 1;

        If (\( O_j \) has a replica in the site) and
            (local_emergency_counter_{kj} = 0) and
            (local_open_read_{kj} < global_open_write_{j})
            Discard the replica of \( O_j \) and inform \( P(O_j) \);
    }
}

Figure 6. The EXIT Algorithm

TIME CHECK Algorithm

When (a local time-check event occurs)
{
    For each local \( U_i \)
    {
        If (the on/off schedule status has changed)
        {
            call EXIT Algorithm for \( U_i \);
            call ENTER Algorithm for \( U_i \);
        }
    }
}

Figure 7. The TIME CHECK Algorithm.
**EMERGENCY ENTER Algorithm**

When (an emergency event occurs at $B_k$)
{
    For all $O_j$ required in the emergency event
    {
        local_emergency_counter$_{kj}$ += 1;

        If (no local replica of $O_j$ exist)
            Allocate a replica of $O_j$ and inform $P(O_j)$;
    }
}

Figure 8. The EMERGENCY ENTER Algorithm.

**EMERGENCY EXIT Algorithm**

When (an emergency event is over at $B_k$)
{
    For all $O_j$ required in the emergency event
    {
        local_emergency_counter$_{kj}$ -= 1;

        If (local_emergency_counter$_{kj}$ = 0) and
            (local_open_read$_{kj}$ < global_open_write$_j$)
            Discard the replica of $O_j$ and inform $P(O_j)$;
    }
}

Figure 9. The EMERGENCY EXIT Algorithm.
Simulation & Comparison

- Algorithms to be compared
- Environment simulation
- Test and comparison
- Conclusion

Algorithms to be compared

- No replication
- Static replication
- Dynamic data allocation
- Adaptive data replication
- Active replication
Environment simulation

- Predefined parameters
  - access class (routinely used objects)
  - location class (routinely used location)

- Variables
  - Movement location (in location class)
  - Access locality (in access class)
  - Schedule conformability
  - Write ratio

Figure 10. Average access cost of five replication schemes with varying movement locality.
Figure 11. Average local availability of five replication schemes with varying movement locality.

Figure 12. Average access cost of five replication schemes with varying access locality.
Figure 13. Average local availability of five replication schemes with varying access locality.

Figure 14. Average access cost of five replication schemes with varying schedule conformability.
Figure 15. Average local availability of five replication schemes with varying schedule conformity.

Figure 16. Average access cost of five replication schemes with varying write ratio.
Conclusion

- Active replication algorithm won the champion in the tournament!
  - Low average access time
  - High data locality
The future work …

To be integrated with other mechanisms:
- Caching
- Prefetching
- Data broadcasting
- etc.

Thank you!