Efficient Barrier Synchronization on Wireless Computing Systems

Nian-Feng Tzeng, Bhanurekha Kasula, and **Hongyi Wu**
Center for Advanced Computer Studies
University of Louisiana at Lafayette

ABSTRACT

This work deals with efficient barrier synchronization for wireless cluster computing where nodes communicate with each other wirelessly in one or multiple hops. Such a computing system is referred to as a wireless computing system (WCS). Given t nodes participating in barrier synchronization on such a WCS of size n ($\geq t$), a *backbone structure* (BBS) is constructed and maintained to cover all the t participants throughout the course of barrier synchronization necessary for correct execution of an assigned task cooperatively. Two mechanisms for BBS construction, dubbed RAD_Expansion and G_Search, have been developed and evaluated empirically, and RAD_Expansion is found to be more desirable. Our transport-layer barrier primitive is then implemented on the BBS constructed using RAD_Expansion, with AODV as its underlying network protocol, for evaluation using NS-2. It arrives at faster barrier synchronization than a naïve approach, which lets every participant send its check-in messages directly to the barrier controller. The barrier time gap widens as the number of participants increases under given mobility.

1. INTRODUCTION

Wireless computing systems have emerged as versatile and capable platforms for handling tasks once deemed possible only by wired, stationary computer systems or servers. This can be evidenced by a new type of resource-sharing networks, called the wireless grid, considered recently [5] for connecting mobile and nomadic devices together to extend grid computing to large numbers of devices that would otherwise be unable to participate and share resources. The wireless grid promises a new type of resource-sharing and distributed computing with various mobile devices involved collectively for speedier completion of a given task on-demand, realized by harnessing the unused cycles of involved devices connected via wireless links. It is thus envisioned that a powerful computing paradigm built upon wireless devices plus wired computers may be obtained to facilitate a pervasive infrastructure for time-starved users. A wireless pervasive infrastructure so constructed is referred to as a wireless computing system (WCS), which will be the focus of this work.

Wireless computing nodes may not be the best choice to constitute high-performance computer systems, when compared with their wired counterparts, due to relatively lower transmission rates and reliability figures associated with wireless links. Nonetheless, there are applications to benefit from a computing paradigm composed of wireless nodes, as the one under consideration here. Transmission rates over wireless links have been improved substantially for the past few years, and they can now (or soon) satisfy the communication needs of moderate- to coarse-grained collaborative computing. Recent research effort began to consider schemes for mobile and wireless computing systems [4], and those schemes help to make such a system possibly become a powerful computing paradigm.

Task execution on a collection of computing nodes usually requires barrier synchronization to ensure correct ordering of operations among participating nodes in regard to producing data and consuming data. Barrier synchronization enforces that all participants have reached a barrier point before any participant may execute beyond the barrier point. As a basic synchronization primitive supported in collaborative execution of tasks, barrier synchronization has been treated considerably in conventional (wired) computing systems. Efficient barrier implementation usually calls for certain logic structures across which activities involved in barrier synchronization are performed.

This article deals with barrier synchronization on a WCS for the very first time. Given t nodes (i.e., hosts) participating in barrier synchronization on the WCS of size n ($\geq t$), a *backbone structure* is constructed and maintained to cover all the t participants throughout the course of barrier synchronization necessary for correct execution of an assigned task cooperatively. Such a structure (denoted by BBS) is established before any barrier operation takes place, and its structure establishment involves all participants in a distributed manner. It calls for self-healing after its links break because of node mobility to ensure correct barrier synchronization. For any barrier structure, a barrier controller knows all the participants, acting to collect check-in messages from all participants when they reach the barrier (known as the check-in process) and to release the barrier after check-in messages of all participants have been received (known as the release process). We have developed two mechanisms for BBS construction, dubbed RAD_Expansion and G_Search, with their results evaluated empirically using NS-2. It is found via our simulation study that RAD_Expansion gives rise to more desirable BBS in terms of longest branch length (which reflects the barrier synchronization time).

The BBS may be built on top of any network-layer routing protocol developed for mobile ad hoc networks, such as DSDV, DSR, or AODV, to obtain a transport-

This work was support in part by the National Science Foundation under Grant CCR-0105529, by the U.S. Department of Energy (DOE) under Award Number DE-FG02-04ER46136, and by the Board of Regents, State of Louisiana, under Contract No. DOE/LEQSF(2004-07)-ULL.

IEEE COMPUTER SOCIETY

layer primitive for barrier synchronization. We have implemented an efficient barrier primitive according to RAD_Expansion, with AODV as its underlying network protocol, for evaluation using NS-2. A large number of barrier operations are carried out under varying amounts of participants for a given WCS. Our primitive so implemented arrives at faster barrier synchronization than a naïve approach, which lets every participant send its check-in message directly to the barrier controller and employs a system-wide broadcast for the barrier release. In addition, the synchronization time difference grows as the number of participants increases under given mobility.

2. RELATED WORK

2.1 Barrier Synchronization

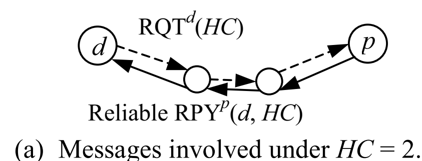
Packet barrier synchronization is common and important to cooperative execution of tasks in computing systems. A barrier participant typically involves three actions during each barrier: (1) posting its arrival at the barrier, (2) waiting for all other participants to reach the barrier, and (3) receiving a notification to proceed beyond the barrier. Previous studies on barrier synchronization considered only wired computing systems. For a wired computing system, efficient barrier synchronization can be achieved by following a logic structure established to cover all participants. In contrast to designating a single node (say, the barrier controller, Γ) for handling the check-in messages from all participants, the structure enables Γ to deal with only a few participants, which are connected directly with Γ according to the structure and which interact with Γ on behalf of their respective connected participants [1]. Such a logic structure remains unchanged throughout job execution after being established, under a wired computing system whose connections among its constituent nodes are static.

No barrier synchronization on the wireless computing system (WCS) has ever been addressed. While previous barrier schemes for wired computing systems establish different logical structures along which barrier participants follow to yield effective check-in and release processes, they are not suitable for the WCS. A desirable barrier implementation on the WCS calls for not only low overhead in creating a logic BBS for participants to follow, but also efficient barrier operations over wireless links, as will be presented in the next section.

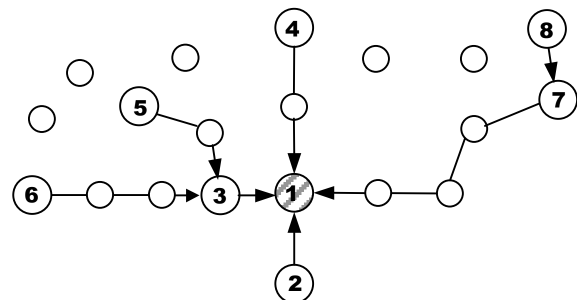
2.2 Leader Election and Mutual Exclusion

Work on leader election for mobile ad hoc networks has been carried out [4], where election algorithms have been devised by modifying TORA (a routing scheme for mobile wireless networks [6]). Unlike TORA, the algorithms require that the node which detects a network partition (due to link breakage because of node moves) elect itself as the leader of the new component before transmitting this information to its neighbors, who in turn, propagate this information to their respective neighbors, etc. [4]. All nodes in the new component eventually will be aware of their new leader. When two or more components meet due to new links established (resulting from node movement), the component leader with the smallest ID becomes the eventual leader of the merged component.

Mutual exclusion involves a group of nodes which coordinately fetch a shared code, known as the *critical section* (*CS*), in a mutually exclusive manner. It is a basic function for a group of nodes to share a common resource properly, allowing only one node at a time to enter CS realized commonly via a lock. Mutual exclusion has been widely applied to solve many problems, and its solutions can be generally classified into two types: permission-based and token-based ones [2]. Token-based solutions are preferred due to lower communication traffic overhead [2]. They are implemented often by first creating a static logical ring over those nodes involved in accesses to the CS and then making a token travel (in sequence or on-demand) along the logic ring so that the node with the token is allowed to enter the CS [2].



(a) Messages involved under $HC = 2$.



(b) Resulting BBS, with N_1 as barrier controller and parent of a participant marked by an arrow line segment.

Fig. 1. Backbone structure (BBS) constructed with RAD_Expansion, where a participant is denoted by a large circle with its ID enclosed, and a non-participant, by a small circle.

3. BBS FOR BARRIER SYNCHRONIZATION

A naïve approach for barrier implementation is to let every participant send its check-in message directly to the barrier controller. In Fig. 1(b), for example, a naïve implementation requires participants N_8 and N_7 to send their check-ins individually to N_1 (the barrier controller) along their respective shortest paths, which happen to share a common segment from N_7 to N_1. This naïve implementation is clearly inefficient because (1) a check-in message may need to use another participant as a relay to reach N_1, but that relay itself uses a separate check-in message (yielding higher traffic without aggregation), (2) given t participants, including N_1, totally *(t-1)* messages are to reach, and be collected directly by, N_1, (3) higher contention for a message to proceed closer to N_1 and thus

IEEE COMPUTER SOCIETY

a longer barrier operation time, and (4) the acknowledgements (ACKs) of a barrier release exhibit an undesirable phenomenon (called *feedback implosion* in reliable multicast [7]) due to many ACKs reaching N_1 simultaneously without aggregation.

Our barrier synchronization primitive is viewed as a transport-layer implementation, built on any network-layer routing protocol devised for mobile ad hoc networks, such as DSDV [8], DSR [3], or AODV [9]. It establishes a backbone structure (abbreviated as BBS) before any barrier operation actually takes place and also maintains the structure throughout the course of barrier synchronization necessary for correct execution of an assigned task cooperatively. We have developed two mechanisms for backbone construction, and the following assumptions are made for all the mechanisms considered here:

- Each node has a system-wide unique ID.
- Each node is equipped with an omni-directional antenna and a transceiver able to communicate up to a fixed range of γ.
- All nodes use the same channel for communications, and all communication links are bidirectional.
- Each node is with low mobility, if any, (i.e., $0 \leq \mu \leq 2$), to reflect the movement of a node owner walking at no faster than 2 meters/sec (or, equivalently, 7.2 km/hr).
- If mobility exists, only a single structural change (e.g., a link break or a link creation) may occur at a time, and the next change may happen only after the whole system has settled fully. Nodes in the WCS always stay within one connected group and are never partitioned into disconnected groups.

Consider a WCS with *n* nodes in total, where a subset of the system nodes is chosen to execute a given task cooperatively, with barrier synchronization involved during execution. If *t* nodes are chosen, known as participants, the BBS constructed includes all the *t* participants, plus possibly some non-participants. The reason for the BBS to possibly contain non-participants is because not every participant has a neighbor which is a participant; if a participant, say N_P, has no participating neighbor, a non-participant node (which is a neighbor of N_P) has to be included in the BBS, so that all participants are involved in a single BBS. Naturally, a desirable BBS should contain as few non-participants as possible to lower traffic overhead during BBS construction and resources involved during barrier synchronization operations. Each participant, other than the barrier controller, has exactly one parent node, whereas the barrier controller has no parent and becomes the *root* of the BBS. Each participant in the BBS after construction, say Node N_P, keeps track of (1) the list of all participants in the substructure rooted at N_P, with those immediate descendents marked, and (2) its parent participant (if N_P is not the root).

3.1 RAD_Expansion (Radial Expansion)

This mechanism relies on a designated node, often the barrier controller, to start the process of BBS construction. The designated node, say N_d, broadcasts a join request message, $RQT^d(HC)$, where *HC* is the hopcount of the message traversed so far, with its initial value set to 0. A neighbor (i.e., node), on receiving $RQT^d(HC)$, responds in a way determined by its role. If the node is not a participant, it simply re-broadcasts $RQT^d(HC)$ after incrementing *HC* by 1. If the node, say N_p, is a participant, on the other hand, it (1) sends a reply back to node N_d (the designated node in this case), $RPY^p(d, HC)$, with *HC* taken directly from received $RQT^d(HC)$, and (2) produces a new message, $RQT^p(HC)$, for broadcasting with its *HC* initialized to 0. Each node accepts only one RQT for processing (no matter which node initiates it) and drops all subsequent ones, if any, ensuring that join request messages always travel radially outward with respect to N_d, dubbed RAD_Expansion. Messages involved between N_d and N_p under RAD_Expansion are shown in Fig. 1(a). After receiving the reply message $RPY^p(d, HC)$, node N_d becomes the parent of node N_p and it also records *HC*, which indicates the number of non-participants along the path from N_d to N_p. The resulting BBS obtained by RAD_Expansion is depicted in Fig. 1(b), where the longest branch length (from N_8 to N_1) is 5.

Any reply message has to be delivered reliably, so that the participant which replies the join message can be added appropriately to the BBS being built. To this end, an $ACK^d(p)$ message is transmitted back to node N_p by node N_d upon receiving $RPY^p(d, HC)$. A timer is set up at N_p when it sends off $RPY^p(d, HC)$, with its time period chosen to be proportional to *HC*. If the timer expires before $ACK^d(p)$ is received, $RPY^p(d, HC)$ is re-transmitted. Note that loops can never be formed during BBS creation under RAD_Expansion. Every time when a new participant is added to BBS under development in this mechanism, it involves just one single node, whereas G_Search may attach a substructure with multiple participants of BBS being built at once. In order to keep the list of participants in a substructure of BBS at its root node (and thus N_d has a list of all participants in the system) after construction is completed, a parent node of each newly joined participant, N_p, sends an update message $UPD(N_p)$ to its own parent upon receiving a RPY from N_p. This update message is forwarded all the way up to the root, N_d, and the transmission of such an update message has to be reliable, using repeated transmissions after the timer expires.

Summary

When N_p receives $RQT^d(HC)$:
    if N_p is a participant, it
      {sends $RPY^p(d, HC)$ reliably;
      broadcasts $RQT^p(HC)$}
    else it adds 1 to *HC* and re-broadcasts $RQT^d(HC)$.

On receiving $RPY^p(d, HC)$, N_p becomes the parent of N_d.

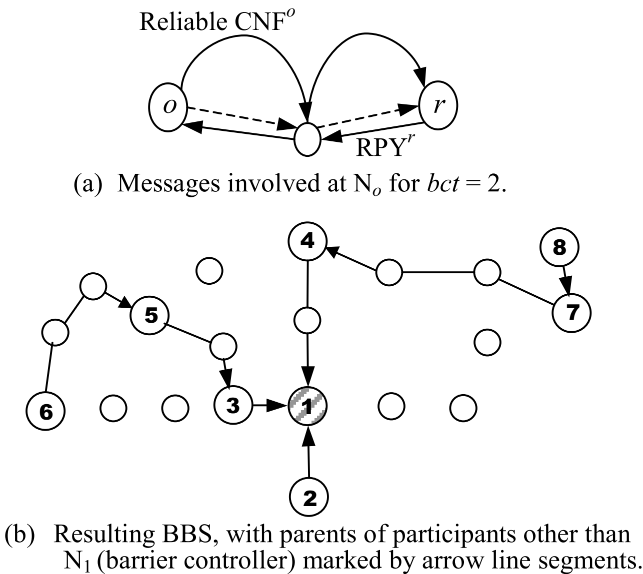
3.2 G_Search

Basic Idea

The idea of this mechanism is to let each participant (other than the barrier controller) search for a nearby participant (as its parent) independently in a distributed manner. After a set of nodes has been chosen for participating in the given task which involves barrier

IEEE COMPUTER SOCIETY

synchronization, G_Search calls for each participant to search for its immediate neighboring participant first, if existing, via a broadcast of a join request message with *TTL* (time-to-live) = 1. Messages involved in this G_Search are (1) $RQT^o(bct, TTL, LST_o)$, a join request message broadcast by (and originated from) a node whose ID is *o*, denoted by N_o, when the broadcast count at the node is *bct*, (2) $RPY^r(bct, o, hc)$, a reply message sent back to N_o from node N_r, and (3) $CNF^o(bct, r, LST_o)$, a confirmation message delivered to N_r from N_o. Each participant, say N_P, keeps a list of all participants resided in the substructure rooted at N_P, denoted by LST_P. This list information is needed to prevent a loop from being formed when participants join the BBS, since LST_o carried in $RQT^o(bct, TTL, LST_o)$ forbids any participant in the list to reply to the request message. The value *hc* is calculated by the replying participant (node N_r) according to its received *bct* and *TTL*, to indicate the number of hops between nodes N_o and N_r. In this figure, node N_o finds its parent node, N_r, which replies to the request message $RQT^o(bct, TTL, LST_o)$ originated from N_o. The value of *bct* at each participant starts with 1, so that search is conducted progressively from each participant to its immediate neighbors first. Each node, upon receiving a request message, always decrements its TTL by 1.



(a) Messages involved at N_o for *bct* = 2.

(b) Resulting BBS, with parents of participants other than N_1 (barrier controller) marked by arrow line segments.

Fig. 2. Backbone structure (BBS) constructed by G_Search.

Detailed Steps

In general, search for a (physical) neighboring participant from a given participant may take place multiple times progressively, with its *bct* increased each time by 1, and the initial *TTL* value of a request message issued from the node is set to that of *bct*. Upon receiving $RQT^o(bct, TTL, LST_o)$, a node replies it using the join reply message, $RPY^r(bct, o, hc)$, if it is a participant and has not received the same $RQT^o(bct, TTL, LST_o)$ before. A node simply drops $RQT^o(bct, TTL, LST_o)$, if (1) it has received the request before, or (2) it is not a participant and *TTL* = 0. Otherwise, the node decrements *TTL* by 1 before broadcasting $RQT^o(bct, TTL)$, no matter whether it is a participant or not. Note that the number of hops $RQT^o(bct, TTL, LST_o)$ takes to reach N_r is given by $(bct - TTL)$, which is put in its reply of $RPY^r(bct, o, hc)$ produced by N_r to notify node N_o that its potential parent N_r is *hc* hops away. As N_o may receive multiple replies, it use *hc* in each reply to decide its preferred parent node.

After N_o originates one $RQT^o(bct, TTL, LST_o)$, a timer, dubbed the join timer denoted by T_{join}, starts to tick at the node (i.e., participant). If no reply message comes back before T_{join} expires, the node originates another join request message. The duration of T_{join} is proportional to *bct*. A join reply message is created by participant N_r upon receiving $RQT^o(bct, TTL, LST_o)$, provided that N_r is not resided in the substructure rooted at N_o (since a loop would otherwise be formed by joining N_o to N_r). After $RPY^r(bct, o, hc)$ is received by node N_o, it is kept in the node if its *hc* is smaller than that of an earlier reply (if any) registered therein; a join confirmation message is produced and sent back to N_r (whose reply is kept in N_o) when the timer at N_o expires. This confirmation message, $CNF^o(bct, r, LST_o)$, carries LST_o to node N_r. The participant N_o has then found its parent (namely, N_r), and they together with the existing backbone substructure where N_r is resided, form a bigger substructure. The path from N_o to its parent is recorded when $CNF^o(bct, r, LST_o)$ travels to N_r. If N_r has a parent, say node N_P, LST_o is to be delivered to N_P for it to update its LST_P via the use of $UPD(LST_o)$ This updating is carried out at every node along the substructure upward starting from N_P, until the root of the substructure is reached. The BBS resulting from G_Search is illustrated in Fig. 2(b), where the longest branch length (from N_6 to N_1) equals 6.

As message collisions happen in any WCS at hand, it is necessary to consider the impacts of message collisions on G_Search correctness. The loss of a RQT or RPY message is tolerable and may cause the join timer T_{join} to expire, forcing another round of parent search. If a CNF message is lost due to collision, however, G_Search breaks. Therefore, a reliable transmission of CNF messages is needed, and it is implemented using an ACK by N_r upon receiving $CNF^o(bct, r, LST_o)$. If N_o fails to get $ACK^r(bct, o)$ for a pre-determined time span (dependent upon its round-trip delay proportional to *bct* and controlled by a separate timer), another CNF message is issued. To reduce the chance of having two or more CNF messages (from different neighboring participants of the same distance away) colliding at N_r, a small random delay is introduced before sending off each CNF.

As the delivery of $UPD(LST_o)$ messages may be delayed or even lost (due to collisions), a loop could exist in the BBS being developed and has to be removed. This is due to an incomplete LST_t at node N_t resulting from delayed or missed update messages when the node attempts to join another substructure. In addition, two (or more) participants may seek each other to be their

IEEE COMPUTER SOCIETY

respective parents at the same time under G_Search, forming a loop. To detect and remove such a loop, node N_P simply examines the received LST_o against its LST_P when receiving UPD(LST_o) before updating its LST_P. If the two lists have no element in common, the update is safe to proceed. If an element is found in both lists, the element is removed from LST_P, with a notice sent to every child of N_P; this same notice will be passed down to all children of a node receiving the notice, if the node contains the element in the substructure it roots. When the element (i.e., a node) is reached eventually by a notice, the element (together with all nodes in the substructure rooted by the element) is trimmed from the BBS under development, making it loop-free. Similarly, if LST_o contains N_P, a loop exists and N_P trims itself from the BBS. Node N_P then repeats its join process, and this time the process succeeds as the participant in its substructure will not reply to the join request.

Summary

When N_r receives RQT^o(*bct*, *TTL*, LST_o) for the first time, it decrements *TTL* by 1 and

if N_r is a participant,

{if it is in LST_o, it rebroadcasts the message

else it sends RPY^r(*bct*, *o*, *hc*) back to N_o}

else it rebroadcast the message if *TTL* > 0.

After receiving RPY^r(*bct*, *o*, *hc*), N_o

{starts a timer to wait for more possible replies, if it is the first reply;

keeps the received reply, if its *hc* is smaller}

When the timer at N_o expires, it sends CNF^o(*bct*, *r*, LST_o) reliably to N_r, whose reply is kept therein.

3.3 Performance Evaluation

Barrier synchronization usually takes place repeatedly involving all the participants in a way governed by the BBS built. To evaluate and compare BBSs built by the two mechanisms described above, we have implemented those mechanisms using the NS-2 simulation platform. This evaluation leads to a preferred BBS chosen for barrier synchronization in WCSs, based on the quality of the resulting structure characterized by such measures as the longest branch length (in terms of hopcounts along the path from the barrier controller to a leaf node) and the BBS size. The first measure tends to dictate the mean barrier synchronization time, while the second measure indicates how many nodes (i.e., resources) are involved in barrier synchronization. Note that a BBS with a smaller longest branch length usually reflects that each internal participant (excluding the barrier controller and leaf nodes) has more children on an average, likely to realize better aggregation of check-in messages to lower communication bandwidth consumed and thus fast barrier synchronization. However, an excessively large number of children with any node in the BBS may cause severe communication contention and thus should be avoided.

Our evaluation was conducted for various numbers of wireless nodes randomly scattered in an area of 800 × 800 meters^2, with the communication range (i.e., γ) of each node fixed to 250 meters. Node mobility (μ) is assumed 0. The MAC layer protocol used is 802.11b and the network layer protocol is AODV [9]. For a given number of nodes, the amount of participants varies during our simulation study for comparing the two construction mechanisms. Simulation outcomes under different numbers of nodes reveal similar trends; thus, only those results for 100 nodes are illustrated here, with the number of participants ranging from 10 to 50. Each result presented in Figs. 3-4 is the average of 30 simulation outcomes.

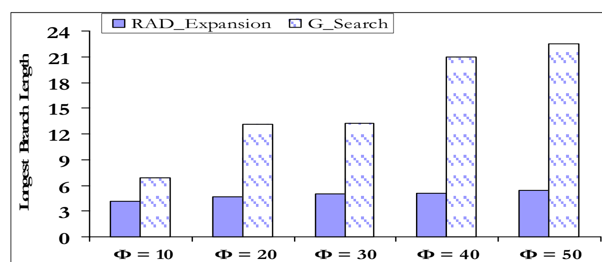


Fig. 3. Longest branch length (in hopcounts) versus Φ.

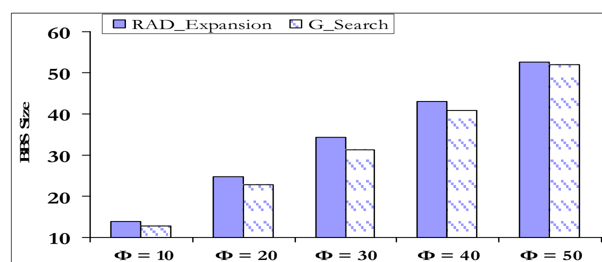


Fig. 4. BBS size versus Φ.

The longest branch length (in terms of hopcounts) versus the number of participants (denoted by Φ) for a WCS with 100 nodes is shown in Fig. 3. It can be observed from the figure that RAD_Expansion leads to relatively constrained longest branch length throughout the range of Φ considered, while G_Search yields a rapid rise in the longest branch length when Φ grows. For Φ = 20, the longest branch length equals 4.7 (or 13.1) under RAD_Expansion (or G_Search); it becomes 5.1 (or 21.0) when Φ is 40 for RAD_Expansion (or G_Search). The BBS size versus Φ is depicted in Fig. 4. Both mechanisms lead to monotonically increased BBS sizes when Φ rises, as might be expected. While RAD_Expansion builds the BBS of slightly larger than G_Search for a given Φ, the size difference is often negligible and always within 10%. Consequently, RAD_Expansion produces a more desirable BBS due to far smaller longest branch length for speedier barrier synchronization. Note that the mean number of children per internal node in the BBS constructed using RAD_Expansion is consistently larger and is always smaller than 5 for all the Φ values evaluated. Therefore, RAD_Expansion is expected to arrive at faster barrier synchronization than its G_Search counterpart due to better aggregation in check-in messages (but without severe

0-7695-2281-5/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

communication contention) and also a more efficient release during the barrier operation.

4. Efficient Barrier Synchronization and Its Evaluation

As RAD_Expansion builds more desirable BBSs, we focus the remaining discussion solely on the BBSs built by RAD_Expansion. Our efficient barrier synchronization follows a BBS, with the check-in process starting from leaf nodes (which are barrier participants) independently and the release process, from the BBS root (which is the barrier controller).

4.1 Efficient Barrier Synchronization

When hitting the barrier, a participant enters the check-in process, which is governed by its position in the BBS created. If the participant is a leaf node, it sends a check-in message instantly to its parent. Otherwise, the participant waits to receive the check-in messages from *all* its immediate descendent participants (called IDPs) before forwarding its check-in message to its parent; it cannot check in until all its IDPs have done so, as explained in our earlier work [1]. After receiving check-ins from all its IDPs, the barrier controller performs a barrier release by notifying all its IDPs using separate release messages. Each IDP acknowledges the receipt of a release message separately, implementing a reliable release. After receiving a release message, an IDP sends the message to every one of its own IDPs separately, with an ACK required from each IDP.

This BBS-based approach eliminates the causes of inefficient barrier synchronization associated with a naïve barrier implementation mentioned in Sec. 3. It should be noted that barrier operations tend to repeat many times during the course of a task execution. Each barrier hence calls for a sequence number, which is held in the barrier controller and advanced by 1 after a barrier operation has completed. The sequence number is carried in a release message, which is sent to the IDPs. Upon getting a release message, a participant checks to see if that release has arrived yet (using the sequence number carried) to avoid redundancy. The receipt of a release at a participant signals that it can proceed to execute codes beyond the barrier point. After spending a certain time period for code execution, the participant hits the barrier again and conducts its next round of barrier operations.

Check-in messages are not transmitted in a reliable way to reduce barrier traffic, thereby fastening barrier operations. To ensure proper synchronization even in the presence of transmission collisions, a timer is maintained at the barrier controller (i.e., the BBS root) and it is initialized to a predefined value (which is proportional to the number of total participants in the BBS) every time when a barrier operation starts. Recall that every participant, say N_P, in a BBS keeps a list of all participants in the substructure rooted by N_P and that those IDPs of N_P are marked, as stated earlier. An indicator is associated with each IDP of N_P to register the arrival of a check-in from the IDP. The indicators of all IDPs at each participant in BBS are reset at the beginning of every barrier operation. The provision of such indicators for IDP(s) at every participant allows the participant to know which of its IDPs have not yet checked in at any point of time. If the barrier controller fails to have all its IDPs check in after its timer expires, it solicits each IDP which has not checked in yet (via a unicast message), to carry out its check-in. This solicitation in turn causes a ripple of downward solicitations all the way down to the node where its check-in has not completed. On the other hand, the release message from a participant requires an ACK from each of its IDPs for a reliable release.

4.2 Self-Healing Maintenance

As nodes with mobility in WCSs may move away from their neighbors in the course of job execution involving barrier operations, communication links established between pairs of nodes in the existing BBS could break, with possibly new links to other nodes created. As a result, a self-healing process via local reconfiguration is employed to re-establish the paths affected by those broken links in the BBS, so that barrier synchronization can proceed successfully. Local reconfiguration involves a fraction of nodes in the WCS (rather than all nodes), and it usually translates to notable savings in traffic overhead and the reconfiguration time.

When a message is to travel over a path affected by a broken link, an unsuccessful transmission is detected and reported by the link layer, which informs the network layer routing procedure, say AODV, of failed message delivery, triggering BBS maintenance for local reconfiguration. Given a failed path between a BBS participant N_χ and its parent N_β, our self-healing maintenance identifies a new path from N_χ to N_β through broadcasting a BBS_REPAIR message, with its *TTL* set to $d+1$ (where d is the number of hops between N_χ and N_β prior to link breakage). The BBS_REPAIR message carries the list of all participants in the substructure rooted by N_χ, and it is re-broadcast by every node (whether participant or non-participant) on BBS exactly once. If it is re-broadcast by any participant in the carried list, a flag is raised in the message (to signify a less desirable choice at N_β). When such a BBS_REPAIR message reaches N_β, a reply is issued immediately if the flag in the arrival message is not raised; otherwise, it is registered and a waiting period starts. In the latter case, a reply is originated only if no other message arrives with its flag unset before the period is over. If no response arrives at N_χ after a preset timer (proportional to d) expires, another broadcast message with $TTL = d+2$ is then originated. This progressive search for a new path to N_β is handled in a way similar to G_Search. Note that a path to N_β through the substructure rooted by N_χ does not cause any loop, since the check-in message from N_χ to N_β is point-to-point delivered; nonetheless, such a path could incur more transmission conflicts and is thus less desirable. After getting a reply from N_β, N_χ stops its progressive search, and any later response arriving at N_χ is dropped.

4.3 Evaluation and Results

The proposed efficient barrier synchronization was implemented as a transport-layer protocol on top of the AODV network protocol [9] under the NS-2 simulation platform for evaluation. The performance measure of interest collected and compared is *mean time* per barrier operation (denoted by ζ). An efficient barrier implementation leads to a small ζ

0-7695-2281-5/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

value. Our evaluation was conducted for various numbers of wireless nodes randomly scattered in an area of 800 × 800 meters^2, with the communication range of each node (i.e., γ) fixed to 250 meters. Each node is assumed to have constant mobility μ meters/sec, with 0≤μ≤2. The following results demonstrate 100 nodes with various numbers of participants, which hit a barrier periodically. Assuming that a single barrier is present in the code, and all participants are involved in every barrier operation. Three node distributions each with five independent runs were conducted in our simulation study for each value shown in Fig. 5. Every participant in each run carries out 200 barrier operations, and the collected results are averaged over all the runs and the node distributions to get every value depicted. If a node has mobility μ > 0, it moves at a fixed speed of μ meters/sec along a direction chosen randomly. The results of a naïve barrier implementation (without following the BBS) are also included in the figure (denoted by "NAI"), where the AODV route discovery process was followed to establish a path directly from each participant to the barrier controller for barrier operations.

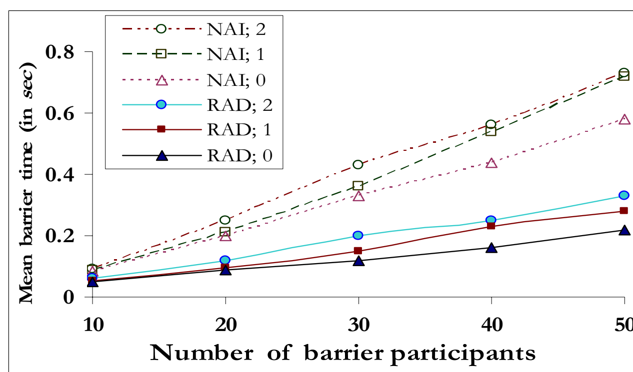


Fig. 5. Mean barrier time (in sec) versus Φ under different mobility μ, with 0≤μ≤2.

From this figure, it is observed that the mean barrier time (ζ) rises slightly as the number of participants increases. This is expected, since higher check-in and release traffic is involved in a larger BBS structure built (according to Fig. 4), as a result of more participants. For a given Φ, ζ moves up as node mobility (μ) increases, because a larger μ leads to higher chances of link breakage and, thus, more frequent self-healing processes (described in Sec. 4.2), which in turn elevates ζ. For Φ = 30, as an example, ζ grows from 0.12 sec under μ = 0 (denoted by "RAD; 0" in Fig. 5) to 0.20 sec under μ = 2 meters/sec (denoted by "RAD; 2" which represents the situation where the owners of wireless nodes are all walking speedily, at 7.2 km/hr). When compared with a naïve barrier implementation, our BBS-based barrier synchronization is expected to be much more efficient for any given μ, as unveiled by the simulation results depicted in Fig. 5. For μ = 1 and Φ = 20, as an example, it takes roughly 0.10 sec on an average to carry out one barrier operation with our proposed approach, in contrast to 0.21 sec following its naïve counterpart. In addition, the curves in Fig. 5 also demonstrate that under any given μ, the barrier time gap (between the proposed approach and its naïve counterpart) increases for a larger Φ.

5. Conclusion

Efficient barrier synchronization has been introduced for the wireless computing system (WCS), realized by constructing a backbone structure (BBS) to quicken synchronization by all barrier participants. Two mechanisms are considered for BBS construction, and RAD_Expansion is found to yield a more desirable BBS. Our barrier synchronization on the WCS is thus governed by the BBS built according to RAD_Expansion. Simulation results obtained using NS-2 have revealed that our proposed approach for barrier synchronization indeed outperforms a naïve method (where each participant conducts its check-in process directly toward the barrier controller and the release process is done via a system-wide broadcast from the barrier controller). Under mobility μ equal to 1 meter/sec, for example, a WCS with 100 nodes and 20 barrier participants takes roughly 0.10 sec on an average for one barrier operation under our proposed approach, as opposed to 0.21 sec following its naïve counterpart. In addition, the barrier time gap widens as the number of participants increases for given mobility.

References

[1] N.-F. Tzeng and A. Kongmunvattana, "Distributed Shared Memory Systems with Improved Barrier Synchronization and Data Transfer," *Proc. 11th ACM Int'l Conference on Supercomputing* (*ICS '97*), July 1997, pp. 148-155.

[2] S. Fu, N.-F. Tzeng, and J.-Y. Chung, "Empirical Evaluation of Mutual Exclusion Algorithms for Distributed Systems," *Journal of Parallel and Distributed Computing*, vol. 60, pp. 785-806, July 2000.

[3] D. Johnson and D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Computing*, Chapter 5, pp. 153-181, Kluwer Academic Publishers, 1996.

[4] N. Malpani, J. L. Welch, and N. Vaidya, "Leader Election Algorithms for Mobile Ad Hoc Networks," *Proc. 4th Int'l Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Aug. 2000.

[5] L. McKnight, J. Howison, and S. Bradner, "Guest Editors' Introduction: Wireless Grids: Distributed Resource Sharing by Mobile, Nomadic, and Fixed Devices," *IEEE Internet Computing*, vol. 8, no. 4, pp. 24-31, July/Aug. 2004.

[6] V. D. Park and M. S. Corson, "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks," *Proc. of Conf. on Computer Communications* (*IEEE INFOCOM 1997*), Apr. 1997, pp. 1405-1413.

[7] S. Paul *et al.*, "Reliable multicast transport protocol (RMTP)," *IEEE Journal on Selected Areas in Communications*, vol. 15, pp. 407-421, April 1997.

[8] C. E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," *Proc. of ACM SIGCOMM '94*, Sept. 1994, pp. 234-244.

[9] C. E. Perkins, E. M. Royer, and S. R. Das, "Ad Hoc on Demand Distance Vector (AODV) Routing," *Internet Draft* draft-ietf-manet-aodv-10.txt, Internet Engineering Task Force, Mar. 2002.

0-7695-2281-5/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY