A COMPUTATIONAL WIRELESS NETWORK BACKPLANE: PERFORMANCE IN A DISTRIBUTED SPEAKER IDENTIFICATION APPLICATION

H. T. Kung, Chit-Kwan Lin, Chia-Yung Su, Dario Vlah
Harvard University
Cambridge, MA

John Grieco[†], Mark Huggins[‡], Bruce Suter[†]
Air Force Research Lab[†], Oasis Systems, Inc.[‡]
Rome, NY

ABSTRACT

A major challenge in the DoD's next-generation network-centric information systems concerns on-demand provisioning of computation and network infrastructures at tactical network edges (e.g., deploying wireless airborne or hybrid air/ground networks). To support this vision, we present DWARF, a general distributed application execution framework for wireless ad-hoc networks which dynamically allocates computation resources and manages failures. DWARF nodes each run a separate task simultaneously, thereby achieving execution speed-up from *parallel processing*. Failed tasks, e.g., due to fluctuating wireless links to mobile nodes, are automatically detected and reassigned, transparent to the application. Further, tasks are executed in an order that satisfies dependencies given by task dependency graphs. To use DWARF, application programmers need only decompose their applications into tasks and define the task dependency graphs.

In this paper, we describe DWARF and report its benefits in running an important existing application—speaker identification—over a 32-node wireless network which supports fault-tolerant computation. We observed two major performance gains: (1) a ten-fold speed-up in identifying speakers due to parallelizing the application, and (2) higher accuracy in speaker identification, made possible by the increased *sensor diversity* provided by geographically distributed nodes. While our nodes have modest computing power individually, combined under DWARF, they are able to execute speaker identification with much greater speed and with improved accuracy.

1. INTRODUCTION

Environments at the tactical edge that require substantial on-demand communication infrastructures pose significant challenges to network-centric information systems. While these areas may be of sufficient tactical interest to provision with such traditional network technologies as satellite communications or mobile base stations [1], these technologies fail to adequately support a rising class of applications at network edges, whose traffic patterns are intense but constrained to a local area. Examples include peer-to-peer applications or sensor data processing in the region. In such cases, the narrow uplinks, which often have other higher priority competing traffic, can only provide service with much smaller bandwidth compared to the capacity of local-area radio channels. By deploying local-area information systems, computation can be pushed to the network edge, allowing data processing and reduction to occur before being transmitted over narrow back-haul links. Alternatively, locally computed results that are locally consumed are immediately available, thereby tightening the decision-making loop in tactical situations. These opportunities suggest that the development of local-area, wireless information systems could yield many advantages at the tactical edge.

But just as the dynamic and possibly extreme nature of such environments hampers deployment of network infrastructure, it also poses additional challenges to the design of such systems. Fluid tactical situations require the ability to provision computation resources on-demand. Battery-powered computation devices and transient fluctuations in link qualities and intermittent network disconnect due to terrain, node mobility, or interference mean such systems must be built with fault tolerance in mind [2]. We believe that distributed system architectures, rather than centralized ones, can satisfy such requirements more naturally, especially as the scale of information systems and application demands at the edge continues to grow.

Traditional high-performance computing systems have far different error characteristics compared to wireless computation backplanes. Their error rates are exceedingly low, owing to designs with high signal-to-noise ratios and redundant encoding. At the same time, the throughputs are higher due to higher signaling rates and indepen-

978-1-4244-2677-5/08/$25.00

dent parallel communication channels afforded by wired interconnection. Achieving high-performance computing over an ad-hoc wireless infrastructure without assuming a traditional wired backplane therefore poses a challenge to system designers. In this paper, we will discuss various techniques, such as those based on wireless broadcast, in addressing these issues.

Furthermore, as a greater number of increasingly capable commercial-off-the-shelf (COTS) wireless mobile devices—replete with sensing components (e.g., GPS and cameras)—find their way into the hands of users and become embedded into the edge environment, new opportunities that deserve exploitation arise. For example, there are increasing numbers of points at which the environment can be sampled, and there is an increasing amount of computation power lurking unused at the network edge. These opportunities suggest that new types of distributed wireless ad-hoc systems could be built which would harness the locally available computation power and apply it to processing of the plentiful sensor data.

More importantly, the growing abundance of sensors in the field permits such systems to exploit *sensor diversity* to enable a class of applications in which high quality sensor data greatly reduces the complexity of the computation required. In some applications, e.g., audio signal processing, even the most sophisticated algorithms cannot adequately process low quality input data. A distributed system, coupled with sensor diversity, presents a natural setting where the highest quality sensor data available can be selected as input into a simpler computation. This is a clear advantage, as only data most likely to yield good results will consume any resources.

New distributed system architectures and tools based on COTS components are needed to properly exploit these opportunities and overcome the challenges outlined above. In this paper we present one such architecture and its prototype implementation: a framework to support building distributed applications on 802.11 wireless ad-hoc networks. We then describe how we applied the framework to a particular local-area application of interest—speaker identification [3]—and evaluate its performance compared to that with a non-distributed version.

2. DISTRIBUTED WIRELESS APPLICATION RUNTIME FRAMEWORK (DWARF)

In this section we describe DWARF, an execution system built to support distributed applications in 802.11 wireless ad-hoc networks. We consider DWARF to be a *computational wireless network backplane*. Conventional wisdom suggests that backplanes connecting parallel processors are required to have high reliability, bandwidth, and throughput; an 802.11 wireless network could hardly fit the bill. However, DWARF's fault-tolerant design mitigates the unreliability of wireless links and its extensive use of wireless broadcasts allows data to be sent to many nodes in one shot (e.g., in input data distribution), thereby compensating for the relatively low bandwidth of the wireless channel in many situations.

We give a detailed description of DWARF in the sections below. Overall, we assume that a node, termed "master," exists at any given time to drive the computation from start to end. This need not be a single physical node, as the participating nodes can monitor the status of the master node and promote a backup node if the master fails.

Note that all wireless transmissions in DWARF are broadcasts; none are point-to-point. We rely on the 802.11 MAC (CSMA) to handle transmission scheduling and channel contention, and assume that all nodes reside in a single collision domain.

2.1. Reliable Distribution of Input Data

Due to the nature of wireless channels multiple receivers can listen to the same physical transmission, a property referred to herein as *broadcast advantage.* Therefore, we have focused attention on problems where the same input data simultaneously serve a multitude of computation tasks on separate nodes. This is more efficient than repeatedly using the channel to independently transmit the input to each node.

Our framework specifies a reliable broadcast mechanism to distribute input data to the nodes participating in a computation. In our implementation, we use a single-hop reliable broadcast protocol based on acknowledgments and retransmissions, structured as follows.

We divide input files of arbitrary size into fixed-size *generations*, each consisting of some number G of link layer packets. The master node then reliably transmits generation by generation until the entire file is transmitted. Within each generation, every T_G seconds the master resends all outstanding packets, where outstanding packets are those that haven't been received by at least one node. Nodes acknowledge each received packet, or in the absence of received packets periodically announce the subset of the current generation they hold. This mechanism ensures that the transmission will make progress even in lossy channels, as long as the probability of success is non-zero.

This reliable broadcasting mechanism is relatively simple compared to some advanced techniques described in the ad-hoc networking literature [4]. We discuss candidate replacement schemes based on network coding in Section 7.

2.2. Task Decomposition

Our framework expects that applications are divided into units of execution termed "tasks". The definition of tasks is up to application writers, but usually they are modules with relatively few dependencies that can run concurrently, thereby achieving parallel speed-up.

The dependency relationships between tasks are described by application writers in dependency graphs. A directed edge between nodes A and B in the graph ($A \rightarrow B$) specifies that task A must complete before B starts, for reasons such as task B requiring input from A, or to impose a certain ordering of side-effects.

In our implementation, task inputs and outputs are opaque to the execution system, and are considered to be files of arbitrary format. In the future we will extend this interface to allow formal data types; with this, the system can inspect application results and perform *task pruning*, an optimization discussed further in Section 7. The output of each task is reliably broadcast to the whole computation group for increased system reliability in the face of node and link failures. Note that since any transmission is likely to be overheard in a wireless medium by nearby nodes, adding a reliability mechanism does not increase the resource demands as much as with point-to-point channels. The reliable output broadcasting mechanism is similar to that used for inputs as described in the previous subsection.

2.3. Distributed Fault-tolerant Task Scheduler

In an environment prone to faults, nodes are not guaranteed to finish their tasks within a predetermined time and may even exit the computation abruptly. Fault-tolerant schemes [5] [6] [7] must be used to ensure all tasks eventually complete. While allowing repeated task execution for fault tolerance, such schemes need to minimize unnecessary redundancy in task execution.

DWARF satisfies these requirements by employing an *optimistic scheduler with fully-replicated control structures* at all nodes. These control structures (detailed below) provide each node in the DWARF system with a view of the global system state and, when kept consistent across nodes, allow the scheduler to assign unfinished tasks efficiently, while minimizing task redundancy. Such a strategy is particularly well-suited to the wireless medium because broadcast advantage significantly reduces the overhead of control structure replication.

Based on this replicated control mechanism, the DWARF scheduler tracks node liveness in order to detect and react to node failure and reassign failed tasks. Each DWARF node runs an instance of the scheduler and periodically broadcasts a heartbeat message. Upon receipt of a heartbeat, a DWARF node records the arrival timestamp in a table, with one entry per node. Nodes are marked as "failed" if subsequent heartbeats are not received after a timeout period. If, at a later time, a heartbeat arrives from a failed node, the scheduler will promote the node back to "live" status.

The scheduler manages task execution by maintaing two data structures: a *dependency graph* with all unfinished tasks and a *priority queue* of ready-to-run tasks (i.e., tasks with completely satisfied dependencies). When the scheduler starts, it performs a topological sort of the dependency graph, resulting in tasks with the fewest dependencies being given the highest priority. This maximizes the number of independent tasks available at any given time and minimizes node idle time.

To minimize redundant work, multiple nodes should avoid scheduling and invoking the same task. To do so, each scheduler broadcasts notifications of task scheduling and completion events and monitors which tasks have been scheduled or completed by other nodes, placing tasks currently being executed by any node at the end of the queue and removing them upon completion. If node failure is detected by the scheduler and results in a task being unfinished, that task is moved back into its original position in the queue.

The scheduler makes use of a *distributed advisory locking mechanism* [8] in order to avoid the situation where multiple nodes schedule the same task at exactly the same time. In this mechanism, node IDs are sorted and placed into a virtual ring topology. In an N-node system, the scheduler on each node adheres to the following lock request policy. A scheduler requests a lock for task t from node i, where $i = t \mod N$. If node i has failed, a new request is made to node $(i + 1) \mod N$ instead. If this too has failed, $(i + 2) \mod N$ is tried, and so on, until a live node is found. Any scheduler receiving a lock request for task t will grant it, provided that it has no outstanding locks on t and has not recently heard another scheduler grant a lock on t. Once granted, a lock must be refreshed periodically as it is automatically released after a certain period of time q. This prevents deadlock when a node holding a lock fails.

Note that each node in the system updates its own lock table when it overhears a lock being granted by another node. As a result, any given node has the most complete picture of current system state possible in the presence of faults. In the case where inconsistencies in notions of node liveness exist, multiple locks for the same task may still be granted by different maintainers, resulting in some redundancy in task execution. However, this is kept to a

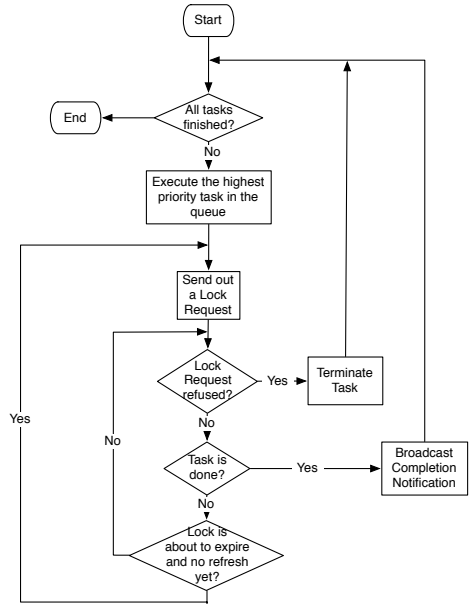

Figure 1: The DWARF fault-tolerant task scheduler algorithm.

minimum by the control structure replication.

The fault-tolerant task scheduler algorithm running on each DWARF node is summarized in Figure 1. On invocation, the scheduler initializes its control structures, and immediately enters the main loop. First, the scheduler checks if all tasks are finished. If so, then all computation is done and the scheduler exits. If not, the scheduler pops the highest priority task from its task queue, immediately begins execution and then sends a lock request. Note that this is why DWARF's scheduler is *optimistic* – it does not wait for a task lock before beginning execution. During execution, the scheduler keeps track of the task's lock status. If the lock is granted, a timer on the lock is started. When the lock is about to expire, but the task has yet to finish, the scheduler refreshes the lock by sending another lock request. If a lock request is eventually refused, the scheduler terminates the task, pushes it onto the tail of the task queue, and returns to the beginning of the main loop. When a task completes, the scheduler broadcasts a completion notification and returns to the beginning of the main loop. On receipt of a task completion notification, the scheduler removes the task from its own task queue.

2.4. Result Aggregation

Our framework assumes that a parallel computation will end at a single node, which collects any pending outputs and combines them into a final result for the entire computation. Due to the broadcast mechanism used for all task outputs, however, it is possible that all nodes that receive a complete set of outputs can play this role. Such nodes broadcast the final result to the network, so that it can be received right away at the node which initiated the computation. In this way, our framework exploits the communication and computation diversity present in the distributed system.

3. DISTRIBUTED SPEAKER IDENTIFICATION: A BENCHMARK APPLICATION

To showcase the capabilities of DWARF, we use open-set speaker identification—a computationally-intensive task—as a benchmark application. Speaker Identification (SID) is the task of determining a person's identity from her voice sample, independent of the words or language she speaks. Typically, SID systems enroll a set of speakers of interest (e.g., authorized personnel). An *open-set* SID system is capable of identifying a non-enrolled speaker as "out-of-set", rather than simply reporting the best "in-set" match. Such systems have military applications including fratricide reduction, enhanced force protection by blue force tracking, or surveillance target identification.

Our open-set SID system is a Gaussian Mixture Model (GMM) classifier based on [3]. Models are trained from Mel-cepstra acoustic features, which capture snapshots of dominant vocal resonances. The first stage of training creates a speaker-independent GMM, called the Universal Background Model (UBM) [9]. Speaker-dependent GMMs for each enrolled speaker are then trained by performing maximum a posteriori adaptation (MAP) of the UBM to a speaker's Mel-cepstra training features. The MAP-adapted speaker models are then finalized via the Minimum Classification Error method, for added discriminative training. For training, we divided the NIST TIMIT corpus [10] of 630 speakers, into two sets: 530 in-set speakers and 100 out-of-set. Of the out-of-set speakers, 50 were included in UBM training. Once 530 speaker models were trained, we scored the models against a separate group of in-set and out-of-set speaker utterances and adjusted a decision threshold value to equalize the number of false acceptances and false rejections.

In the recognition phase, feature vectors are extracted from an unknown speaker's utterance and scored against each speaker model. If the maximum normalized score is greater than the decision threshold, then the unknown speaker is declared to be in-set and her identity is the speaker whose model produced that maximum score. Otherwise, the unknown speaker is declared to be out-of-set.

This recognition computation grows linearly with the number of enrolled speakers and is computationally

intensive—with 530 speaker models, computing the result for one 5-second audio clip on a single 1GHz node in our wireless testbed takes 11 seconds on average. Our goal here is to significantly speed up SID, and we do so via parallel processing. A simple and natural decomposition of this problem is to divide the database of speaker models across a set of computation nodes. For example, if there are 10 nodes, we divide the 530 speaker models into 10 disjoint subsets, each with 53 speaker models. When presented with a speaker utterance, each node is responsible for scoring it against a different speaker model subset in parallel. Subsequently, each node reliably broadcasts its calculated scores to all other nodes. Once a node has received all 530 scores, it can perform adjudication locally.

We have implemented a distributed version of SID as described and have deployed it on DWARF. Using the experiment scenario and system setup described in Sections 4 and 5, we present performance measurements under various conditions in Section 6.

4. EXPERIMENT SCENARIO FOR THE DISTRIBUTED SPEAKER IDENTIFICATION APPLICATION

Consider a set of 16 microphone sensors arbitrarily placed in a square area of 10-ft unit length. Each sensor is connected to a computation node that is equipped with a 802.11b/g Wi-Fi network interface, modest computing resources and is a participant in DWARF. At an arbitrary location within this area is a single, stationary speaker (i.e., no co-channel speech present), whose sound pressure level is 30dB at the source. At a second arbitrary location is a stationary audio noise source, emitting pink noise (10dB at the source). Nodes do not know their own locations, nor those of the speaker or noise source. For our experiments, we have chosen speaker, noise source and sensor locations as shown in Figure 2.

Each node is responsible for monitoring its acoustic neighborhood for speech. We assume that each node digitizes and segments its audio stream into homogeneous speaker segments but, due to the arbitrary locations of the speaker and noise source, each node may capture such segments with varying signal-to-noise ratios (SNRs). The open-set SID algorithm will perform poorly when presented with a speech segment that has low SNR. However, with multiple nodes distributed at various geographical locations, we are able to exploit sensor diversity by first determining which node has the highest SNR and using only that node's speech segment as input for all the other nodes.

Specifically, our distributed SID method adheres to the following task pipeline:

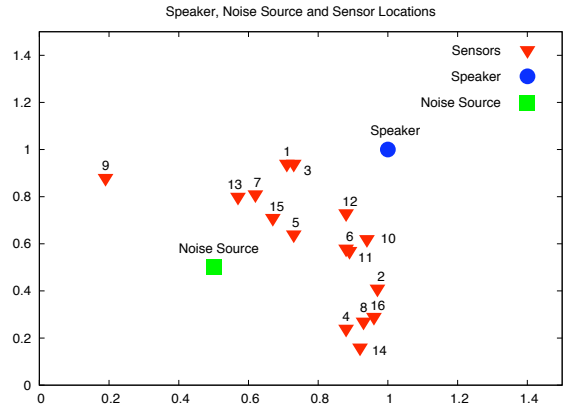

Figure 2: Speaker, noise source and sensor locations in our experimental configuration. The speaker is located at (1,1) and the noise source is located at (0.5, 0.5). Each of the 16 sensors, denoted by triangles, is shown at its location in red.

1) Each node calculates the SNR of the first 250ms of the speech segment generated by its own microphone.
2) All nodes participate in a distributed SNR election that determines the node with the highest SNR.
3) The election winner calculates a speaker feature matrix from its input speech segment and reliably broadcasts this matrix to all other nodes.
4) Upon receipt of the feature matrix, each node scores the input against its subset of speaker models in parallel. When complete, the resulting score vector is reliably broadcast to all other nodes.
5) When a node has aggregated all the score vectors, it performs adjudication on the scores and returns the SID result.

By deploying the application on DWARF, we demonstrate: (1) significant reduction in execution time; (2) fault tolerance against node failure; and (3) ability to support on-the-fly addition of mobile nodes to speed computation. We describe these results in the rest of the paper.

5. EXPERIMENTAL WIRELESS SYSTEM SETUP

We deployed DWARF and our distributed SID application on a wireless ad-hoc network of 32 nodes (Figure 3). Each node is a 1GHz VIA C7 processor with 1GB RAM, a 1GB flash drive, and a USB 802.11b/g wireless network interface (VIA chipset), running Ubuntu Linux 7.04. We have modified the VIA Wi-Fi driver such that IBSS beaconing is disabled and variable rate broadcast is enabled. For all our experiments, we use 11Mbps modulation for broadcast packets.

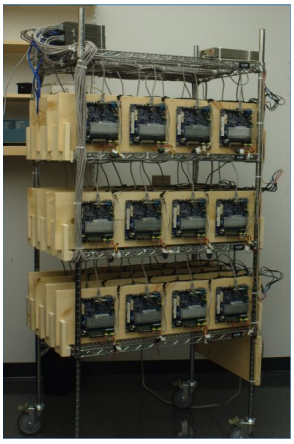

Figure 3: Our wireless ad-hoc network consists of 32 computation nodes, each running DWARF. Given the area (10ft $\times$ 10ft) considered in our speaker identification application scenario, rack-mounting the nodes instead of placing them in the configuration as shown in Figure 2 represents a reasonable approximation in terms of link quality and contention characteristics.

For practical reasons, we simulated the noisy speech segment heard at each of the 16 sensor locations (see Figure 2) by adding pink noise to a noise-free sample. The amount of noise added reflects the expected SNR at each sensor location. Thus, the input into the distributed SID application running on each DWARF node is a WAV file containing a simulated noisy speech segment.

The computation nodes we use in our experiments were not geographically distributed as shown in Figure 2, but were instead rack-mounted as shown in Figure 3. We have previously verified that the configuration of nodes on the rack produces link qualities that are very similar to nodes placed at random within a 10ft $\times$ 10ft area. While simplified, this configuration still employs a true wireless environment in which to measure DWARF performance. In contrast, had we instead emulated wireless channel characteristics over wired Ethernet, it would have been difficult to achieve the timing accuracy required to accurately model collisions.

Note that while the experiment scenario specifies 16 simulated sensors, our testbed consists of 32 computation nodes. This gave us the opportunity to increase the level of parallelism during computation by simply connecting up to two computation nodes to each sensor.

To stress our system, we required a larger speaker model set than the TIMIT corpus could provide. We artificially inflated the size of the speaker model set by duplicating each model, thus requiring the distributed SID application to compute scores for 1060 speaker models (530 unique).

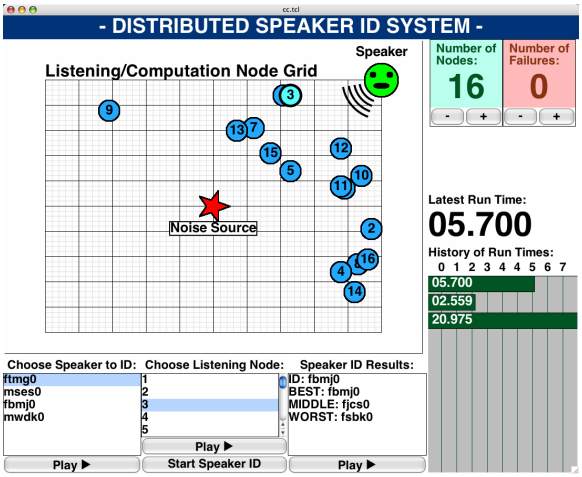

Figure 4: Graphical front-end for managing our distributed speaker identification application on DWARF.

We have also implemented a graphical front-end (Figure 4) to control the distributed SID application. The front-end allows us to pick the number of nodes to use in the computation, play the audio as spoken by the speaker and the noisy audio heard at each of the 16 sensor locations, launch the distributed SID application, record the overall running time, and inspect the returned results by playing a representative speech segment of the speakers corresponding to the top-, middle- and bottom-scoring models. For fault-tolerant experiments, the front-end also allows us to pick the number of nodes to fail. With this graphical console, we have a convenient way of checking the correctness of the DWARF system in executing the distributed SID application.

6. PERFORMANCE RESULTS AND DISCUSSION

Using the experiment scenario and setup described above, we evaluated our system according to three major criteria: (1) speed-up due to parallelization, especially under faulty conditions; (2) the amount of system overhead resulting from DWARF; and (3) the speed-up due to mobile nodes rejoining a computation after a period of departure.

6.1. Speed-up Due to Parallelization

Since one major goal in our work is to reduce the execution time of speaker identification, we are interested in measuring the speed-up due to parallel processing on DWARF. To do so, we measured the average running time (20 trials) of the distributed SID application, as shown in Figure 5. Note that since all nodes listen for a complete set of score vectors before performing local score adjudication,

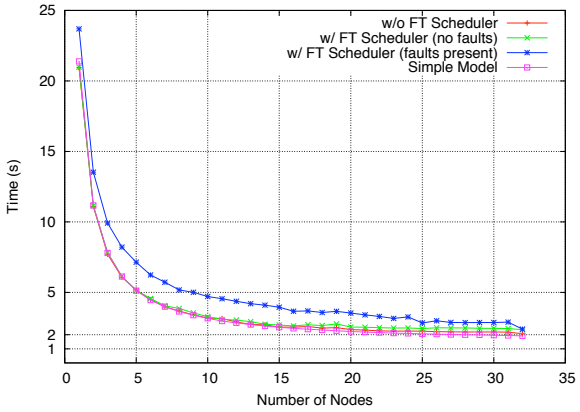

Figure 5: A comparison of average running time (20 trials) of distributed speaker identification over DWARF, under different conditions.

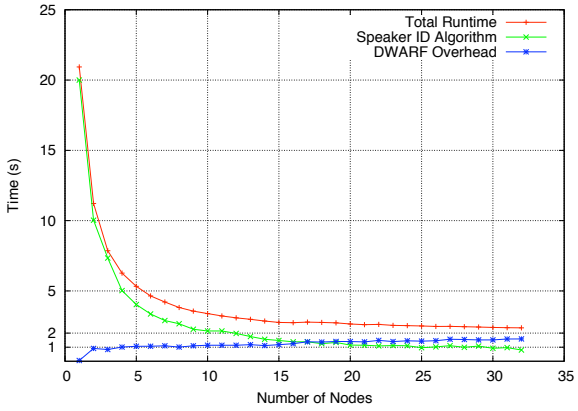

Figure 6: Average running time (20 trials) of distributed SID over DWARF with a varying number of nodes and the FT scheduler enabled, but with no faults present.

transient fades in the wireless links can result in different adjudication start times. This, in turn, leads to different completion times. As a result, we define "running time" as the time elapsed until the first adjudication result is reported by any node.

We compare three experiment conditions to a simple simulation model (purple line) in which we take into account our implementation overhead (audio input pre-processing, timeouts) and data transmission time under 10% packet loss. The simple model represents the optimal speed-up that can be achieved with our implementation in the absence of node failures (i.e., no faults present).

Our baseline measurement in the experiments is the running time of the distributed SID application over DWARF without the fault-tolerant scheduler (see Section 2.3) in the loop or any faults. To show speed-up, we vary the number of nodes in the computation group from one to 32 (red line). Next, we establish that the fault-tolerant scheduler does not introduce significant overhead. To do so, we performed the same experiment, but with the fault-tolerant scheduler handling the invocation of the computation tasks (green line).

Finally, we measured system performance under faulty conditions (blue line). Starting with 32 nodes in the computation group, we inject synthetic faults by terminating the fault-tolerant scheduler on k randomly selected nodes immediately after it has broadcast its first scheduled task. For the blue line in Figure 5, the x-axis represents the number of nodes that remain alive after fault injection (i.e., $n = 32 - k$). Since the system initially starts with 32 nodes, the computation is decomposed into 32 tasks; as faults occur, the scheduler detects which tasks were interrupted and reassigns these to run on the nodes that remain in service. In this way, we demonstrate that DWARF successfully recovers and drives the computation to completion. However, because there are fewer working nodes, we expect the average run-times to increase with the number of faults and indeed, we observe this behavior.

6.2. Overhead in the DWARF System

In all three cases, the results in Figure 5 show an approximately ten-fold speed-up when using 32 nodes, as compared to using a single node. While we show that our implementation matches well with our simple model, the speed-up grows only marginally beyond ∼20 nodes and is short of being linear. Investigating further, we decomposed our running times into their two major constituent parts—the SID computation (green line) and DWARF overhead (blue line)—as shown in Figure 6. We can see that the DWARF overhead (blue line) eventually begins to dominate the SID computation time (green line), even though we still observe a marginal speed-up as the number of nodes increases. This is because the rate of change of overhead is lower than that of the computation time of a task.

In general, the cost of reliably broadcasting data to every node over lossy wireless links is the time the sender requires to detect a lost packet at the receiver(s) and to retransmit. This cost rises with the number of nodes because, with more nodes, the probability that any one node loses a particular packet increases. Retransmission efficiency decreases in tandem, further increasing overhead. For example, if only 1 out of 32 nodes lost the packet, then its retransmission is useless to the other 31. Meanwhile, as we increase the number of nodes, the task size per node decreases as we decompose the overall computation into smaller units. The overall effect is that the computation-to-I/O ratio decreases, implying a reduction in parallel speed-up. One possible method for alleviating this general problem is to relax the condition of reliable broadcast.

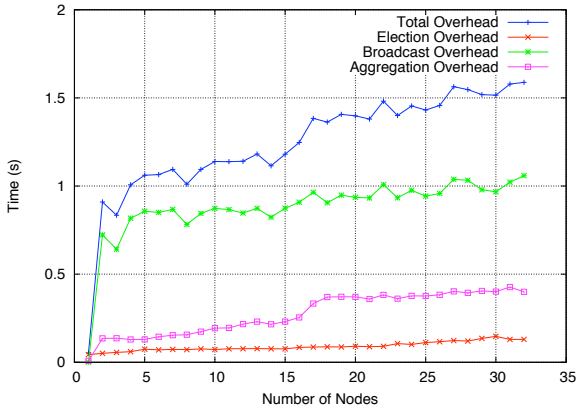

Figure 7: A breakdown of average overhead (20 trials) incurred by distributed speaker identification over DWARF.

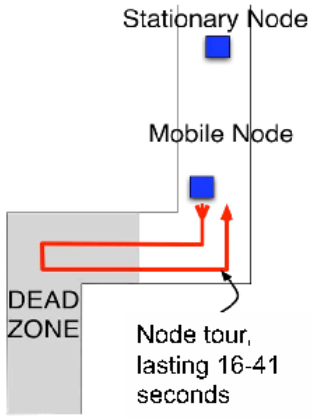

Figure 8: A diagram of a simple mobility experiment. The red line indicates the tour route of the mobile node. The gray area, labeled "dead zone", indicates the region in which the mobile node can no longer communicate with the stationary node.

For instance, if 90% of participating nodes completely receive the data being broadcast and immediately start computation, they may return a result more quickly than if they wait for the remaining 10% to finish receiving. We will explore this trade-off in the future.

Figure 7 further decomposes DWARF overhead into its constituent parts, allowing us to identify the specific I/O bottlenecks. It is important to note that the election overhead (red line) remains low as the number of nodes increases, meaning that the cost of exploiting sensor diversity is quite low.

In contrast, the speaker feature matrix broadcast time (green line) and result aggregation time (purple line) increase. The broadcast time experiences a 1.4-fold increase from two nodes to 32 nodes; aggregation time shows a 2.9-fold increase over the same range. While both are partially due to the increasing number of nodes contending for the channel, differences in computation task completion time also contribute to the increase in aggregation time. By definition, result aggregation cannot complete until the last remaining task finishes. With 32 nodes, we have observed as much as a 250ms difference between the first and last task completing, suggesting that the actual amount of time spent communicating results is comparatively small and that much of the overhead can be attributed to waiting for the last result to become available.

6.3. Results for Mobile Nodes

Finally, we show that DWARF can support node mobility. Consider a scenario where an unmanned aerial vehicle (UAV) loaded with computation nodes participates in a distributed computation with nodes on the ground. The UAV may become connected or disconnected to the ground nodes, depending on its current location. We demonstrate that DWARF can support and take advantage of this depart-and-return mobility model. In this experiment, we employ two indoor nodes running distributed SID on DWARF, as illustrated in Figure 8. The computation is divided into 32 tasks and, initially, the two nodes are within range when the computation starts. Subsequently, one node moves along the tour circuit shown, departing from communications range and then returning. We vary the duration of the mobile node's tour and show that, upon return, the node is automatically re-incorporated into the distributed computation by DWARF. Note that in this single experiment, we address two distinct mobile scenarios: node exit and node entry. The latter case demonstrates DWARF's ability to add mobile computing resources on-the-fly.

Figure 9 shows the average running time (3 trials) of our SID application across different tour durations. First, we present two control measurements which supply the bounds to the running time: with a single node, the job takes 23.7 seconds to complete (upper bound; pink bar); with two nodes, it takes 13.5 seconds to complete (lower bound; green bar). The blue bars represent the running times across various tour durations, and show that running time decreases with faster tours (i.e., earlier return of the mobile node). This demonstrates that, upon return, the mobile node was able to help speed the computation. Note that even for the longest tour (41s), where the mobile node returned after the stationary node had finished the entire computation, we still experience speed-up. This is because the mobile node remains in range at the beginning of the tour and is able to complete some tasks and report their results before moving out of range.

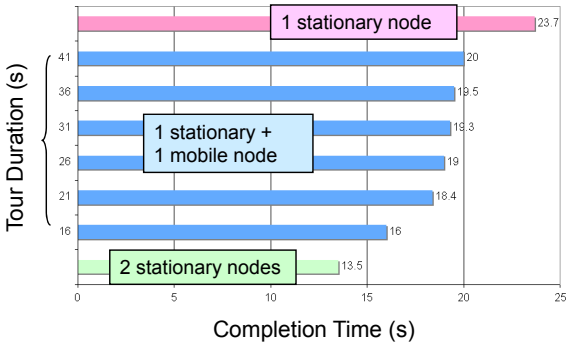

Figure 9: Average running time decreases with faster tours of the mobile node in Figure 8. The mobile node helps the computation even after it returns from beyond communication range.

7. FUTURE WORK

There are several areas of work we plan to address in the future. First, the reliable broadcasting primitives used by our implementation leave room for efficiency improvements in two ways. First, the protocol overhead can be reduced by using an advanced coding scheme such as network coding, which, according to recent results by Ghaderi et al. [11], provides on the order of $\log n$ reduction in transmission attempts with a broadcast group size of n. We have done experiments demonstrating the use of network coding in speeding up content distribution over ad-hoc wireless networks [12]. Secondly, in some cases the full reliability of broadcast may be unnecessary—say, when only a subset of nodes is used to provide a degree of redundancy or parallelism for an operation.

The DWARF framework currently assumes the set of tasks to be completed is static. This means that all of the tasks making up an application are computed by the distributed system. However, in some cases, once an application arrives at some result, it becomes unnecessary to complete any remaining tasks; therefore, a mechanism taking advantage of wireless broadcast is needed whereby an application can quickly communicate such stopping conditions to the runtime system in order to let it *prune* any redundant work scheduled for execution. For example, in the aggregation phase of our speaker ID application, a node will not need to broadcast its scores when it has heard better scores from other nodes. In some parallel computing problems such a mechanism could reduce the running time by orders of magnitude, leading to superlinear speed-up.

Further work is needed on managing the computation group memberships, that is, discovery of the set of nodes that participate, arrive, and leave from a computation. Protocols to address this issue would be based on neighborhood discovery schemes along with heuristics to determine when poorly connected nodes ought to be shed from the computation group.

An important, little-explored avenue of future work involves the use of location information in distributed computing. For example, node locations obtained via a GPS receiver available on mobile nodes or a multi-sensor geolocation scheme could be used to track mobility patterns and then form more reliable computing groups. Other uses could involve location-aware scheduling, where node cluster structure could drive the assignment of tasks to nodes such that small cuts of the task dependency graph fall between well-connected node clusters.

There are several areas in the security arena that need to be addressed, distinguished by the type of attack they defend against. To protect against eavesdropping on the system, all protocol communications would need to be encrypted. Padding would be needed to defend against traffic analysis. Routing schemes that employ detailed wireless channel state information could help resist jamming attacks, in case that the jamming only affects a part of the network. Lastly, a group authentication scheme is needed to admit new nodes to the system.

The task scheduling algorithm of DWARF should be expanded to take into account additional resource constraints, such as battery power, CPU power, or communication bandwidth. For example, it might be desirable to place a heavier load on nodes with more energy reserves, or an external power source, and omit nodes whose energy reserves fall below a certain critical threshold.

Finally, we plan to expand our experimental activities to a testbed with a larger number of mobile handheld nodes, such as the recent MID devices based on the Intel Atom platform [13]. Such devices are small enough to support interesting application scenarios, while still equipped with powerful wireless computation networking components such as several radios, GPS, cameras, and touch screen interfaces.

8. CONCLUSION

In this work, we present a Distributed Wireless Application Runtime Framework (DWARF) and demonstrate its capabilities via an example distributed speaker identification application. Our major contributions are as follows.

First, we have shown that such applications, when deployed on DWARF, can experience significant speed-up due to parallelization. In general, applications that require data broadcasts are particularly well-suited to our framework, since DWARF's wireless backplane utilizes the medium's broadcast advantage to reduce I/O overhead

in data distribution. Further, we have demonstrated that 802.11b/g at 11Mbps is more than sufficient to support compute-bound applications such as the distributed speaker identification problem considered in this paper. Additionally, application porting or development on DWARF is simple for application programmers, requiring only task binaries and a task dependency graph be supplied.

Second, DWARF-based applications automatically gain the fault-tolerant capabilities inherent to the framework. DWARF's fault-tolerant task scheduler enables computations to complete in spite of node failures (e.g., resulting from battery depletion in mobile nodes or traveling out of radio range) while making efficient use of the available parallelism to decrease running time. DWARF recovers from transient failures of wireless links resulting from fading and radio interference by automatically detecting task interruption. Moreover, DWARF can reassign tasks to other nodes upon node departure and re-tasking nodes upon return. The fault-tolerant scheduler accomplishes all these while honoring task dependency conditions and giving priority to critical tasks whose completion will allow a relatively larger number of tasks to be run in parallel in the future.

Third, and most importantly, we demonstrate two important interactions between our architecture and the algorithms it can support. First, the distributed nature of DWARF naturally lends itself to algorithms where sensor diversity can be exploited. In our distributed speaker identification implementation on DWARF, the SNR election plays a critical role in calculating accurate results. Without leveraging this advantage of sensor diversity, even the most cleverly-designed signal processing algorithm would be ineffective if sensor data with very low SNR were used as input data. DWARF is architected such that applications can easily capitalize on such diversity. Second, sensor diversity coupled with local wireless ad-hoc networking enables nodes to share sensor data and to quickly make local decisions without the need for communicating with a centralized arbiter over a slower, possibly unreliable or unavailable, long-haul connection (e.g., satellite). This is particularly advantageous to applications at the tactical edge, where computation results must be quickly produced and are typically consumed locally. Where data transmission over a long-haul link is necessary, DWARF can be used to pre-process and reduce the data, making more effective use of such narrow, shared uplinks.

ACKNOWLEDGMENTS

This research was supported in part by the Air Force Research Laboratory Grants FA8750-08-1-0220 and FA8750-08-1-0191. We also thank VIA Technologies, Inc. for contributing its C7 processors used in our wireless testbed.

REFERENCES

[1] R. North, N. Browne, and L. Schiavone, "Joint tactical radio system - connecting the gig to the tactical edge," in *MILCOM*, 2006, pp. 1–6.

[2] C. Dion-Schwartz, "Networked Communication Capabilities: a Proposed FY08 New-Start Research Program," Airborne Networks Technical Review, 2007.

[3] P. Angkititrakul and J. H. Hansen, "Discriminative in-set/out-of-set speaker recognition," in *IEEE transactions on audio, speech and language processing*, vol. 15, no. 498–508. IEEE, 2007.

[4] B. Williams and T. Camp, "Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks," in *3rd ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, Lausanne, Switzerland, Jun. 2002.

[5] D. Mosse, R. Melhem, and S. Ghosh, "Analysis of a fault-tolerant multiprocessor scheduling algorithm," *Twenty-Fourth International Symposium on Fault-Tolerant Computing*, Jun 1994.

[6] R. Guerraoui and A. Schiper, "Software-based replication for fault tolerance," *Computer*, vol. 30, no. 4, pp. 68–74, 1997.

[7] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *OSDI*, 2006.

[9] V. Prakash and J. H. L. Hansen, "A cohort - ubm approach to mitigate data sparseness for in-set/out-of-set speaker recognition," in *INTERSPEECH 2006 - ICSLP Ninth International Conference on Spoken Language Processing*, September 2006.

[10] J. Garofolo, L. Lamel, W. Fisher, J. Fiscus, D. Pallett, N. Dahlgren, and V. Zue, *TIMIT Acoustic-Phonetic Continuous Speech Corpus*. Linguistic Data Consortium, Philadelphia, 1993.

[11] M. Ghaderi, D. Towsley, and J. Kurose, "Network Coding Performance for Reliable Multicast," in *IEEE MILCOM*, October 2007.

[12] C.-M. Cheng, H. Kung, C.-K. Lin, C.-Y. Su, and D. Vlah, "Rainbow: A wireless medium access control using network coding for multi-hop content distribution," *MILCOM 2008*, Nov. 2008.

[13] Intel Corporation, “Intel Atom Processor,” http://www.intel.com/technology/atom/index.htm.