Adaptive Spatiotemporal Node Selection in Dynamic Networks

Pradip Hari, John B. P. McCabe, Jonathan Banafato, Marcus Henry, Ulrich Kremer,

Dept. of Computer Science, Rutgers University

Kevin Ko, Emmanouil Koukoumidis, Margaret Martonosi,

Princeton University

Li-Shiuan Peh

MIT

PACT'10, September 11–15, 2010, Vienna, Austria

Outline

- Introduction
- Spatiotemporal Properties
- Sample Applications
- Solution Framework
- Experimental Evaluation
- Conclusion

Introduction

- A dynamic network is a set of potentially mobile devices (nodes) in a specific geographic area, which forms spontaneously rather than being configured in advance.
- Devices offer services to each other, (eg. sensing, accelerated computation, and data storage).
- Each device must share its resources (e.g., I/O device access, CPU time, disk space, etc.).
- Uses of such networks :
  - Traffic monitoring,
  - Distributed search and surveillance
  - Grassroots launching of new mobile services
  - Community- based participatory research
- Unlike a traditional sensor network, the nodes of a dynamic network may all have different owners who seek compensation for the use of their resources.
- Assume a virtual currency system to incentivize service provision; units of currency are called “**credits**".
- Nodes advertise a price in credits for each service they offer, which they can change in response to demand or resource scarcity (such as battery charge level).
- Applications purchase services, and must stay within their budgets.

Introduction contd..

- Motivation:
  - Dynamic network applications have spatiotemporal properties that can be exploited to achieve a better overall program outcome.
- Lead to a novel strategy for expressing and satisfying an application's node selection needs.
- **Sarana** framework used
- What is **Sarana**?
  - A high-level parallel programming architecture for dynamic networks
  - Enables applications to express their spatiotemporal properties,
  - Uses these to perform effective, automated node selection and scheduling under a user-specified budget.
  - Properties can be holistic, describing the desired spatial or temporal distribution of selected nodes.
  - They can also be dynamic, describing desired run-time *events* (computational results or sensor readings), so that Sarana can target devices likely to yield such events.

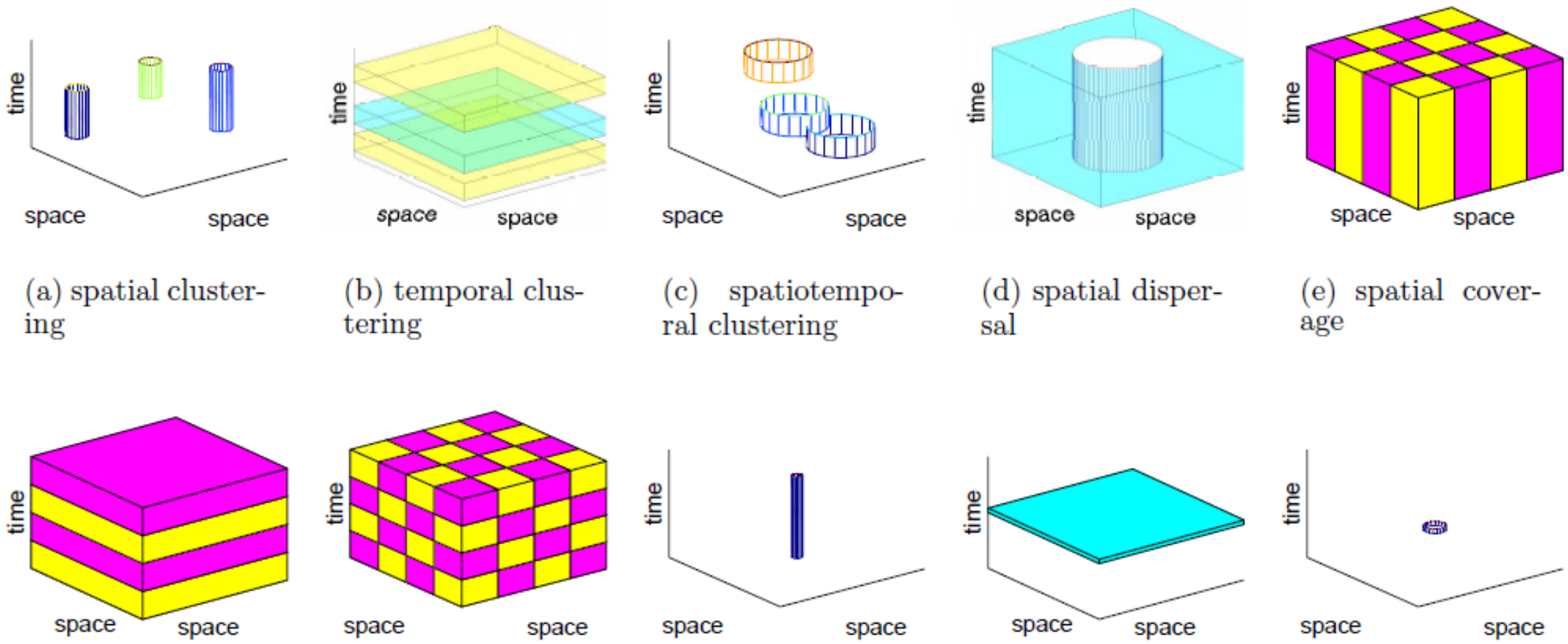
Contributions:

- Language:
  - Employs a parallel loop construct to describe distributed computations.
  - Each iteration visits a node and invokes a service.
  - Allows to express spatiotemporal properties of the application and constructs
  - Application can evaluate the results of a distributed computation and provide dynamic feedback
  - Chose a language over a library implementation of our new programming abstractions
    - Allow a simpler specification of spatiotemporal properties, enable compile-time analyses and future optimizations.
- Prototype:
  - Sarana implementation consists of
    - Compiler,
    - Run-time system,
    - Adaptive space-time aware scheduler
- Three driver examples
  - a collaborative image capture and analysis application (Amber Alert),
  - a straight forward sensing application (Bird Tracking),
  - a space and time-dependent image sampling application (Crowd Estimation).

Spatiotemporal Properties

- **Clustering:** Desirable events may exhibit spatial and/or temporal locality.
  - In spatial clustering node selection is directed towards promising geographic sub-regions.
  - In temporal clustering node visits are increased in the immediate aftermath of a positive event.
- **Dispersal:** The presence of one event at a particular time and place may make it unlikely that another desired event would occur nearby.
  - In spatial dispersal, node selection is directed away from unpromising geographic regions
  - In temporal dispersal, node visits are discouraged in the immediate aftermath of a negative event.
- **Coverage:** An application may need a representative sample of sensor readings over a geographic region or time period.
  - With spatial coverage, a geographic region is divided into equal-area sectors and each sector is sampled (if possible).
  - With temporal coverage a time period is divided into equal intervals and a sample is taken at each interval (if possible).
- **Synchronization:** An application may need a set of sensor readings at the same time or place.

Spatiotemporal Properties



(a) spatial clustering

(b) temporal clustering

(c) spatiotemporal clustering

(d) spatial dispersal

(e) spatial coverage

(f) temporal coverage

(g) spatiotemporal coverage

(h) spatial synchronization

(i) temporal synchronization

(j) spatiotemporal synchronization

Figure 1: Spacetime patterns that can be expressed in Sarana. Shading indicates spacetime subregions that should be preferred in node selection.

Sample Applications

- **Amber Alert** (spatiotemporal clustering).
  - Amber Alert is a program run by U.S. law enforcement agencies to find abducted children, particularly during the crucial first few hours after the abduction has been reported.
  - If a search target is identified, the image is sent to participating smartphone and PDA users near where the image was taken.
- **Bird Tracking** (spatial dispersal, weighted events).
  - This application tries to record bird songs in a target area (e.g.a forest) using a network of microphones.
- **Crowd Estimation** (temporal synchronization, spatialcoverage).
  - This application tries to take photos that will be used to estimate the number of people at a large outdoor gathering.

Solution Framework

- **Language Constructs**
- Sarana adds network macro-programming extensions to the Java programming language.
  - A spatial region is an abstract set of devices observing a specific service in a geographic area.
- ***visit*** statement is similar to a parallel FORALL loop : loop iterations may be executed in parallel on separate nodes of the dynamic network.
  - ***visit*** loops do not contain synchronization or data dependences other than those generated by reduction variables.
  - A loop body may contain report statements, which signal user-defined events.
- Events describe the outcome of individual loop iterations
  - Raising an event will correspond to the determination that an iteration produced useful results.
  - A floating-point value between zero and one indicates the weight of the event, 1.0 being the strongest possible feedback.
- Clauses to specify spatiotemporal heuristics to guide node selection and scheduling.
  - Clauses can have parameters; most clauses take an event name and a radius.
- At least one spatial and one temporal heuristic will be employed.
  - Default strategy is to choose locations and times where iterations can be executed at low cost.
- A Sarana service is represented by a persistent Java object which is accessible to the calling program through the variable declared in a ***visit*** statement

Sarana working

- **Compiler**
  - The Sarana compiler is based on *javaparser*,
  - performs a source to source translation from Sarana to Java.
  - A Sarana application program is divided into independently schedulable code segments called *tasks*.
  - The current task division scheme creates a distinct task for the main program and the body of every visit, and places each task into its own Java class (the *task-class*).
- **Program Execution**
  - **Run-Time System Components :** The Sarana run-time system is a persistent background process that runs on every Sarana-enabled node, and includes several components.
    - The ***directory*** is a network-wide system that maintains a map of Sarana-enabled nodes, their geographic locations, and their available services and service costs.
      - It currently consists of a central server and client modules on each node,
      - To be converted to a distributed, peer-to-peer model in the near future.
    - The ***policy controller*** restricts the use of a node's resources by setting prices for the node's services, according to an administrator defined configuration.
    - The ***service manager*** instantiates and supervises Sarana services installed on a device.
    - The ***process manager*** supervises the execution of individual tasks received from injection nodes.
      - It monitors tasks to ensure that they do not exceed their cost budgets.
    - The ***code distribution manager*** handles up- load of Java class files to remote execution sites.
    - The ***system call handler*** provides common services to local applications only (e.g., obtaining GPS location or currently remaining credits).

Execution

- Execution begins on the injection node with the main task.
- When a visit loop is reached, the run-time scheduler is invoked, tasks corresponding to the body of the loop are distributed to the selected nodes, and the system waits for results to be returned.
- Intelligent scheduling of visit loops is accomplished through the use of *incremental execution*.
  - When executing a visit loop on which a cluster-space or disperse-space clause has been specified, Sarana first conducts a *probing pass*.
  - The desired spatial region divided into several equal-sized sectors, and a suitable node in each sector is selected. Execution of the loop provides partial results and event reports.
  - Based on these reports and the spatiotemporal heuristics chosen for the visit, an expected value is assigned to each node in the space
- Sarana then constructs a schedule for spending the remaining credits allocated to this loop.
- The schedule is a tree which describes the set of nodes to be visited during this loop, the number of credits to be spent at each, and sub-schedules for each nested loop to be initiated at each visited node.
- The schedule construction is an optimization problem where every node has an expected cost and an expected value.
- To avoid wasting credits, a node will not be selected for a particular loop if its geographic location does not make sense.
- **Schedule Construction.** Sarana's schedule optimizer employs a greedy heuristic
  - 1. Query the directory for *targets, e*xclude targets that have been visited on recent probing passes.
  - 2. Construct a minimum-cost plan: a tree containing just enough targets at each nesting level
  - 3. For each remaining target, compute a value density: the ratio of the target's expected quality contribution to its service cost.
  - 4. Insert the targets into a priority queue ordered by decreasing value density.
  - 5. Remove potential targets from the queue and consider them for inclusion in the plan.

Sample Code

```
SRDecl ::= spatialregion space = service @ region ;
ReportStmt ::= report event [= weight];
VisitStmt ::= visit Range Member Timeout
        [: Heuristics] { Stmts }
Range ::= [ (minNodes, maxNodes) ]
Member ::= provider in space
Timeout ::= [ by timeout ]
Heuristics ::= Heuristic {, Heuristic}
Heuristic ::= Spread | Random | Sync
        | CtrlSpace | CtrlTime
Spread ::= spread-space | spread-time
Random ::= random-space | random-time
Sync ::= sync-space | sync-time
CtrlSpace ::= (cluster-space | disperse-space) Ctrl
CtrlTime ::= (cluster-time | disperse-time) Ctrl
Ctrl ::= (event [, distance])
```

Figure 2: Syntax specification.

```
1  spatialregion cameraSpace = CameraService @ Circle(150); // cameras within 150 m of injection
2  sum_reduction int displayedCount = 0;
3  collection_reduction ArrayList<ImageBlob> foundSet = new ArrayList<ImageBlob>();
4  // at least 1 camera, events cluster within 20 m and 1 sec, overall deadline is 30 sec
5  visit camera in cameraSpace by 30 : cluster-space(GOODPIC, 20), cluster-time(GOODPIC, 1) {
6     ImageBlob image = camera.takePhoto();
7     spatialregion analysisSpace = AnalysisService @ Circle(150); // analysis nodes within 150 m of camera
8     or_reduction boolean success = false;
9     visit (1,1) analyzer in analysisSpace { // exactly 1 analyis node
10       success |= analyzer.analyze(image); }
11    if (success) {
12       report GOODPIC; // quality obtained from this execution
13       foundSet.add(image);
14       spatialregion displaySpace = DisplayService @ Circle(30); // displays within 30 m
15       visit (1,10) display in displaySpace {
16          display.show(image);
17          displayedCount += 1; } } }
```

Figure 3: Amber Alert code skeleton.

```
1  spatialregion microphoneSpace = MicrophoneService @ Circle(1000);
2  collection_reduction ArrayList<SoundBlob> foundSet = new ArrayList<SoundBlob>();
3  visit microphone in microphoneSpace : disperse-space(NOISE, 50) {
4     SoundBlob sound = microphone.recordSound();
5     if (isNoise(sound)) {
6        report NOISE = sound.getVolume(); // stay further away from louder noises
7     } else if (isBird(sound)) {
8        foundSet.add(sound); } }
```

Figure 4: Bird Tracking code skeleton.

```
1  spatialregion cameraSpace = CameraService @ Circle(1000); // cameras within 1 km of injection
2  collection_reduction ArrayList<ImageBlob> foundSet = new ArrayList<ImageBlob>();
3  visit (100,120) camera in cameraSpace by 300: spread-space, sync-time {
4     ImageBlob image = camera.takePhoto();
5     foundSet.add(image); } }
```

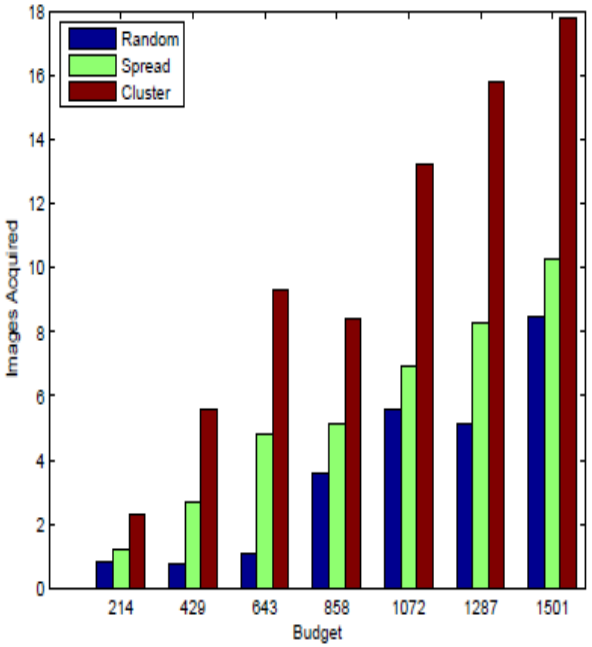
Figure 5: Crowd Estimation code skeleton.

Experimental Evaluation

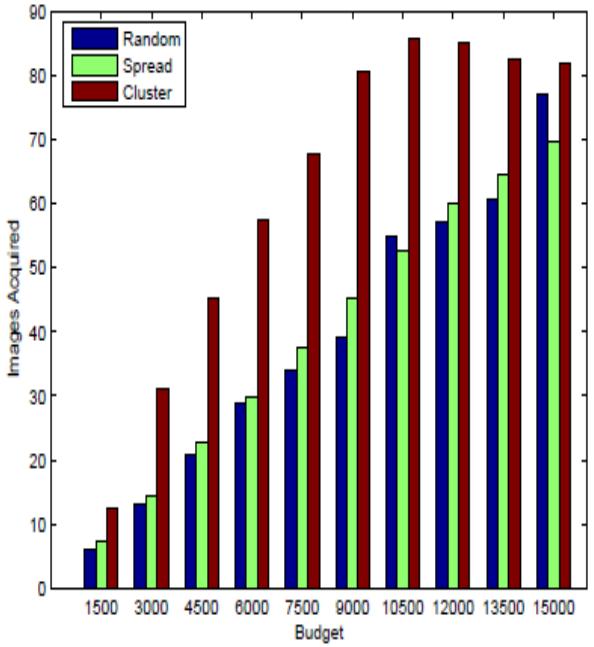
- **Physical Prototype:**
- Sarana runtime system installed on 11 Nokia N810 tablet PC devices, 14 Neo FreeRunner (Openmoko) smartphones, and one Apple Macintosh laptop running OS X 10.5.
- N810 and FreeRunner both have GPS receivers. All devices have microphones.
- The N810 has a built-in camera
- Camera service simulated on the FreeRunner, returns predefined image.
- For lack of real birds, input to the microphone was simulated.
- **Simulation environment:**
- Simulations performed on a cluster of 79 Dell workstations running Fedora Red Hat 4.1.2 Linux.
- A master control program distributes tasks to each workstation and monitors their output.
- Each physical machine runs multiple instances of the Sarana run-time system, representing separate nodes in a dynamic network.
- At start, Master server reads a configuration file which describes the spatial region in which the trial will be conducted.
    - This file describes the set of devices present, the services provided by each, and the initial locations of each.
    - Services relying on physical hardware- camera or microphone- simulated.
    - Nodes offering the camera service given stock images that will be returned when a photo is “taken"
        - Image returned varies with time
        - Each camera can take a photo up to once per second.

Results & Evaluation

- Spatiotemporal Clustering: Amber Alert



(a) clustering, physical



(b) clustering, simulation

Figure 6: Amber Alert with spatial and temporal locality, with and without adaptive scheduling.

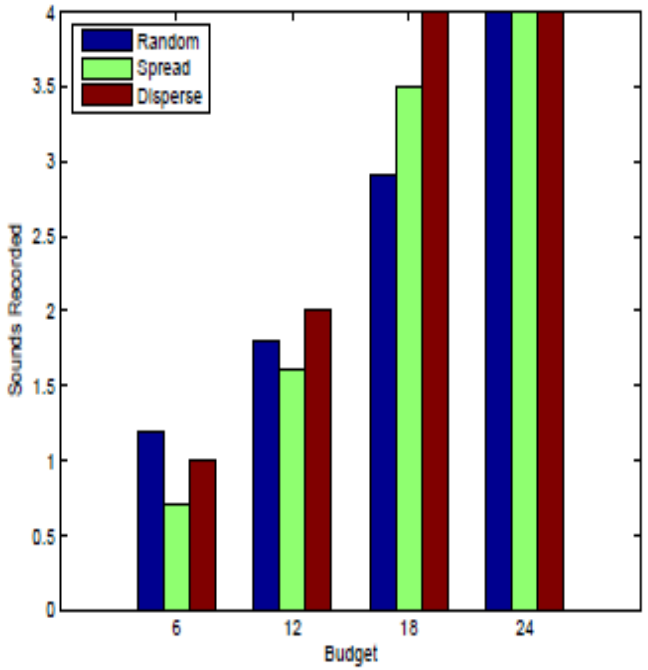
Physical trials:
25 randomly placed camera/display nodes and one injection node.
cost of each service : camera, 10; analysis: 2; display, 5
Experimental trial lasts for 30 seconds overall.
Target subjects visible for three seconds or less, starting at a randomly selected time.
9 such subjects : possible to obtain 27 photos of these
Exhaustive exploration : taking a photo with every camera as frequently as was possible, and trials were performed with 5% (214) to 35% (1501) of the budget necessary to do this.
For each budget, a trial was run 10 times, and the results of these 10 trials were averaged.

Simulation:
90 randomly placed camera/display nodes, 9 analysis nodes, and one injection node.
10 such subjects, and it is possible to obtain 90 images
Trials were performed with 10% (1500) to 100% (15000) of the budget necessary to exhaustively explore the space. For each budget, a trial was run 30 times in each of 5 randomly generated configurations, and the results of these 150 trials were averaged.

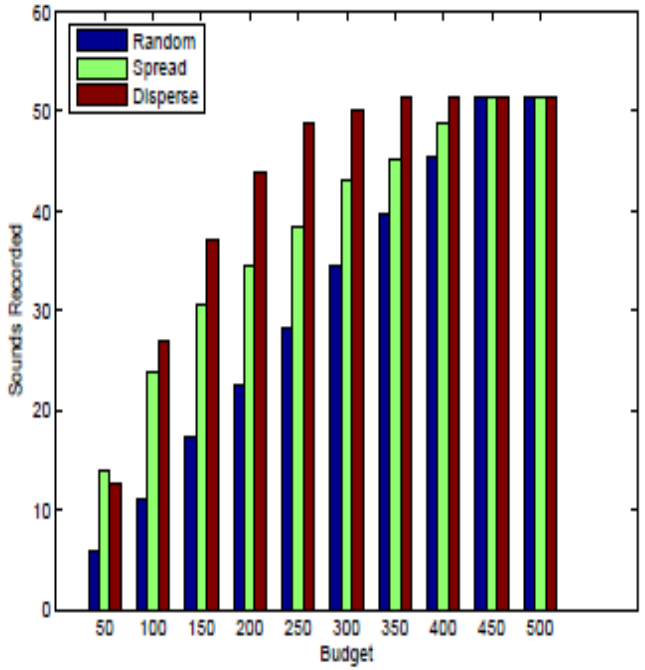
Performance Improvement : up to 745% measured, 117% simulated

Results & Evaluation

- Spatial Dispersal and Weighted Events : Bird Tracker



(a) dispersal, physical



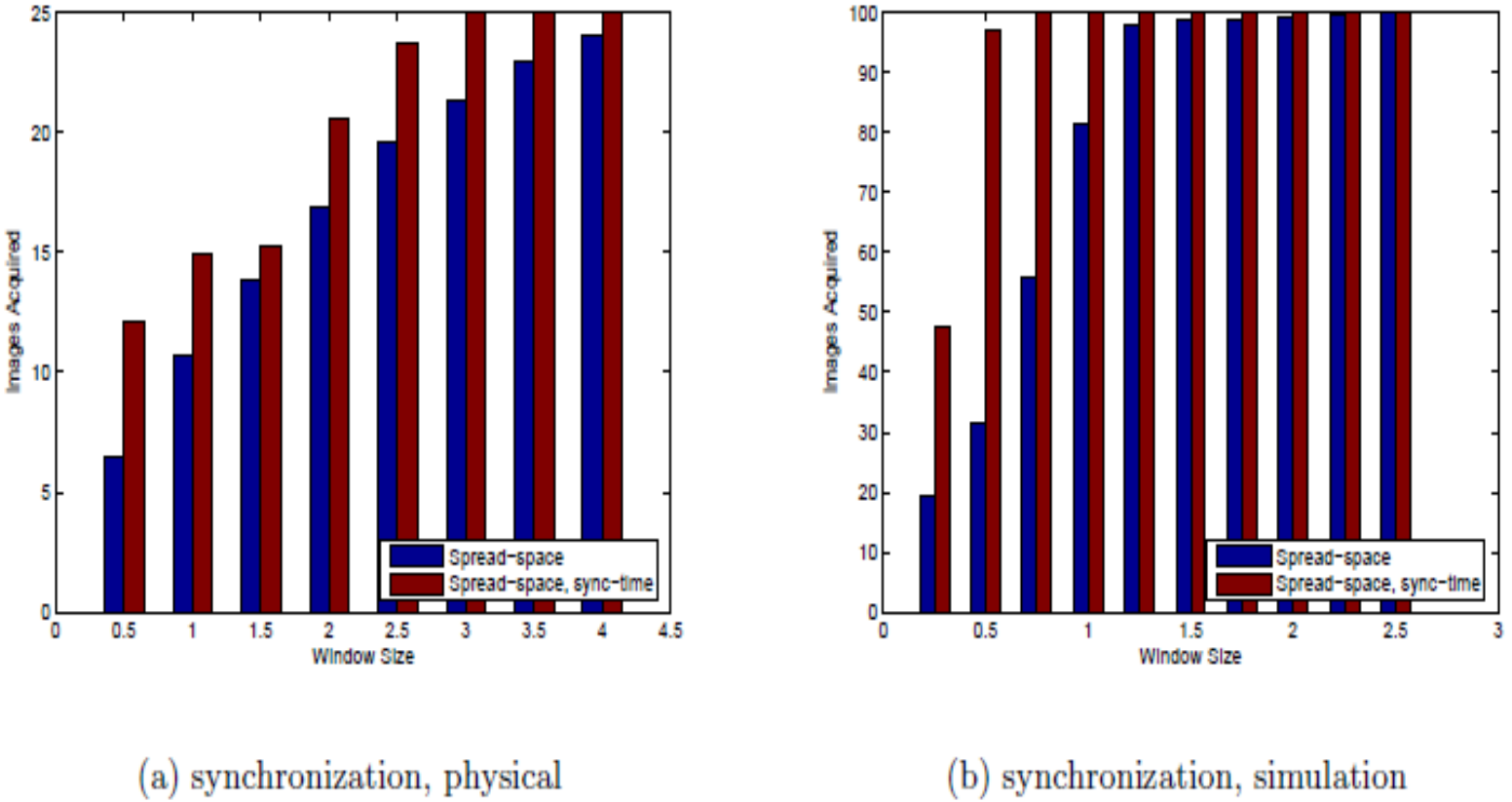
(b) dispersal, simulation

Figure 7: Bird Tracking with spatial dispersal, with and without adaptive scheduling.

Performance Improvement:
38% measured
142% simulated

Results & Evaluation

- **Spatial Coverage and Temporal Synchronization : Crowd Estimation**



(a) synchronization, physical

(b) synchronization, simulation

Figure 8: Crowd Estimation with temporal synchronization, baseline algorithm (spread-space only) and sync-time. Graphs show the maximum number of photographs timestamped in any time window of the given size.

Performance Improvement:
86% measured
209% simulated

Conclusion

- Performance of resource-constrained dynamic network applications can be significantly improved by exploiting their spatiotemporal properties.
- This improvement can take the form of better outcomes for the same resource budget, or comparable outcomes at much lower cost.
- Sarana is the first system that allows a user to specify spatiotemporal heuristics at the language level that are harnessed at run time to adaptively schedule the execution of an application on a set of mobile nodes.

Future Work

- Investigating possible spatiotemporal optimizations and their impact on program performance.
- Privacy and security are clearly important issues in any mobile network environment where resources might be shared.
- Resource-constrained systems will require tradeoffs between the degree of security and the resources required to maintain that degree of security.
- Framework can be extended to allow the programmer to express security preferences effectively