Analysis and Selection of a Scheduling Algorithm for Control of a CubeSat-Class Open Source Spacecraft

Evaluation of Multiple Algorithms

Calvin Bina, Jeremy Straub
Department of Computer Science
University of North Dakota
Grand Forks, North Dakota, USA
calvin.bina.2@my.und.edu, jeremy.straub@my.und.edu

***Abstract*—Effective task scheduling is critical to spacecraft operations. In this paper, the process of developing a task scheduling algorithm for a CubeSat class satellite is detailed. Several algorithms are compared and evaluated using a simulation environment. The results of these simulations are analyzed and the utility of using each scheduling algorithm for various applications is determined and presented.**

Keywords—scheduling; algorithm; spacecraft; CubeSat; open source; evaluation

I. INTRODUCTION

The OpenOrbiter project at the University of North Dakota is working to develop a set of designs for a CubeSat class software as well as a working, modular collection of open source code that could be used by other CubeSat projects as a starting point for development. The availability of these designs and codebase should foster accelerated development for other CubeSat projects, allowing those projects to focus their effort on their own application area, instead of reinventing the proverbial wheel. One aspect of this is to implement a task scheduler which will run on a Raspberry Pi flight computer unit in the satellite and will be responsible for scheduling tasks that will be performed by the satellite. These tasks will include assignments that will be uploaded to the satellite from ground communications station as well as tasks that will routinely check the status of the satellite.

Tasks will be given a priority level to aid the scheduler in determining which sequence of tasks to run. System status checks, for example, will generally have a higher priority than other tasks to ensure that the satellite is still operating correctly. Some tasks, such as those that specify a location to photograph, will only be considered to be scheduled for applicable points in time. At all other times, they will have an infinitely low priority that prevents them from being selected. An expected task run time is calculated for each task, which is used to terminate tasks that have exceeded their allocated time.

The input for the scheduling function will consist of the set of all tasks that need to be run. After applying the scheduling algorithm to that set to find a subset consisting of an optimal sequence of tasks, that subset will be returned to the main program for execution.

Multiple types of schedulers, based on techniques commonly used for computer scheduling are being considered for satellite scheduling herein, including: First Come, First Serve — where the first tasks added to the queue of tasks to be run are the first to run, Earliest Due Date — where the task with the earliest due date is the first to run, Shortest Processing Time — where the task that takes the shortest amount of time to run is the first to run, Critical Ratio — where the heuristic involves computing (time until task due date)/(task processing time) for each task, Slack per Remaining Operations — where the heuristic involves computing (time a task can be delayed and still finish by its due date)/(number of remaining operations) and Round Robin — where each task is given an equal amount of time to run, regardless of whether or not it completes.

The efficacy of each type of scheduler is assessed relative to the goal of having a time and resource efficient scheduling algorithm that ensures successful spacecraft operations and maximizes the performance of tasks relative to performance constraints their respective due dates.

II. BACKGROUND

A. Small Spacecraft and CubeSats

Small spacecraft are as old as space exploration itself: the first spacecraft, Sputnik, was by most definitions, a small spacecraft [1]. More recently, interest has turned to small spacecraft and – in particular – CubeSats due to the increasing mission capabilities that they are able to offer, resulting from electronics miniaturization. The CubeSat form factor was developed in the early 2000s, based on the OPAL [2] design, to facilitate incorporating hands-on development into engineering courses [3].

Unlike larger spacecraft, which may cost millions to design and launch, a CubeSat can be designed from scratch for $250,000 or purchased from a vendor for under $50,000 [4]. Low-cost launch services for this form factor are also available [5, 6], as are free-to-developer launches for qualified institutions [7]. Initially considered by some as being little more than toys [3], CubeSats' utility have been shown for education [8-10], technical development [8, 11] and scientific work [12, 13]. They have even been proposed for use on interplanetary missions [14, 15]. The Open Prototype for Educational Nanosats (OPEN) design [16, 17] has been created to further reduce costs, by allowing a 1-U (10 cm x 10 cm x 11 cm, 1.33 kg) CubeSat to be developed for a parts cost of under $5,000. Its software system [18] and early work on its scheduling [19] have been previously described.

B. Software Systems

Computational capabilities of CubeSats are growing. One design [20], for example, incorporates multiple onboard GumStix computer-on-module processors. Another [21] proposed a distributed architecture, using an I2C bus to separate operations and mission functionality, based on functionality separation presented in [17]. Brandon and Chapin [22] propose development using ADA for CubeSat software; however, Fitzsimmons [23] suggests that rapid development is facilitated by using open-source software, instead.

The Jet Propulsion Laboratory [24] has demonstrated significant CubeSat autonomy, including for "instrument processing and product generation". Many missions, historically (see [25]), have utilized Earth-based planning (in some cases with extensive human involvement and review) as a safeguard to prospective mission problems. In fact, the so-called "seven minutes of terror" of the Mars Science Laboratory (more commonly known as Curiosity) [26] was due to having to rely on onboard autonomy during entry, descent and landing.

III. Algorithms

This section provides an overview of the several common algorithms that are compared in this paper. Each of the following was implemented and considered as a prospective algorithm for spacecraft control scheduling. Note that one key change to the standard implementation of these scheduler, as they are typically used for process scheduling, is that items will only be scheduled into a period that they can be performed in (i.e., an image capture task will only be scheduled when over the target area), if applicable.

A. First Come, First Serve

The first come, first serve algorithm is arguably one of the simplest scheduling algorithms to implement. A ready queue of tasks that are waiting to be run is maintained. Each new task that is sent to the satellite is added to the end of the queue. Due to its simplicity, this algorithm is comparatively fast at scheduling tasks. It is also fair, as the only task characteristic taken into consideration is the task arrival date. However, if a certain task runs for an arbitrarily long amount of time, other tasks may be delayed.

B. Earliest Due Date

This algorithm takes the due date of each task into consideration when creating an execution schedule. This is a simple calculation, making the algorithm both fast and easy to implement. However, the run time of a task is not taken into consideration. If a long running task was scheduled and executed, other tasks may not be able to finish by their due date, causing them to take priority over other tasks.

C. Shortest Processing Time

The shortest processing time algorithm is also fairly simple. The task from the set of tasks waiting to be scheduled with the shortest expected run time is scheduled next. This minimizes the average waiting time for a task in the ready queue. However, multiple problems could arise when putting this algorithm into practice. Any due date a task may have is ignored. This could cause longer running tasks to become starved if there is an abundance of short running jobs.

D. Critical Ratio

A Critical Ratio algorithm takes into account both the due date for a task and the amount of time needed to complete a task. Calculating a tasks critical ratio value takes more time than the work required for the previously mentioned algorithms, but this does help ensure tasks are completed by their due dates. However, if a task is not completed by its due date, that task takes priority over other tasks in the queue. This could cause further delays.

E. Slack per Remaining Operations

This algorithm is similar to the Critical Ratio algorithm in that a value for each task is calculated and that value is used to schedule tasks. To calculate that value, the number of remaining tasks left to be run is taken into account as well as the amount of time remaining for a task to start its execution and still finish by its due date. This algorithm is slightly more complicated than the others mentioned above, but it recognizes important task attributes and responds accordingly.

IV. Evaluation

After implementing each of the forgoing algorithms, a simulation program is used to test them using multiple test data sets. The test data sets will be generated with a separate program which will allow for different distributions to be used to generate a data set. The configurable test data generation program facilitates the creation of data sets that contain different distributions of tasks to simulate scheduler behavior across multiple expected real-world situations.

The performance of each of the separate algorithms will be measured using several metrics. Although the names of the algorithms are reminiscent of CPU-scheduling algorithms, the performance metrics are necessarily different. An optimal task schedule for the CubeSat will ensure that all available tasks are completed by their due date and tasks with geospatial limitations are executed at the appropriate time. Additionally, the algorithm should include *scheduling failure resistance* by scheduling tasks as densely (as early in the scheduled period) as the tasks allow (allowing time for re-performing tasks, if necessitated). The schedules created by each of the algorithms are compared to an optimal schedule (created using an exhaustive task placement and evaluation method) and the level of deviation is reported.

The resources and power required to run the scheduler are also of extreme importance. Given this, the duration of scheduling is also recorded for each scheduler and compared to facilitate the relative comparison of schedule optimality versus speed and cost.

V. Results

Eight different task types were defined within the test data generation program. The attributes of these task types were modeled to reflect multiple maintenance type tasks, a ground communications type task, and multiple tasks types for taking and processing pictures. Uniform frequency distributions were used for maintenance tasks, communications tasks, and picture processing tasks, while the Normal distribution and the Chi-Squared distribution were used for picture taking tasks.

The test data generation program provides options for specifying the overlap of certain attributes of generated tasks. One option clusters the start times of tasks closer together. This makes multiple tasks eligible to be scheduled at any point in time, making the scheduling algorithms work harder and reveal distances between themselves. Another option sets the due date times of generated tasks to be much closer to their respective start times. This makes it impossible for the schedulers to schedule every task by its due date; however, some schedulers deal with this problem more effectively, as shown from the results.

Four test data sets with a length of 200 tasks were generated with the aforementioned combinations of start time and due time options. The averages of the results are shown below.

The results are quantified by the following statistics: average finish time, time utilization, average number of tasks waiting to be scheduled, and average job delay. The average finish time is the average time a task takes from its start time to the time it is done processing. The time utilization is the percentage of time the satellite is processing tasks. This metric is generally very low due to gaps of time between start times of tasks. The average number of tasks waiting to be scheduled is self explanatory, and the average job delay is the average amount of time each job was delayed.

Here are the results:

No start conflicts, no due time conflicts:

First Come First Serve

Average Finish Time: 5886.300000

Utilization: 0.008014

Average # of Tasks: 124.789061

Average Job Delay: 299.690000

Shortest Processing Time

Average Finish Time: 1667.160000

Utilization: 0.013847

Average # of Tasks: 72.218324

Average Job Delay: 23.645000

Earliest Due Date

Average Finish Time: 2365.900000

Utilization: 0.011659

Average # of Tasks: 85.767627

Average Job Delay: 15.470000

Critical Ratio

Average Finish Time: 2325.780000

Utilization: 0.010661

Average # of Tasks: 93.800363

Average Job Delay: 4.205000

Slack per Remaining Operations

Average Finish Time: 1763.870000

Utilization: 0.013371

Average # of Tasks: 74.787789

Average Job Delay: 0.000000

No start conflicts, due time conflicts:

First Come First Serve

Average Finish Time: 9040.010000

Utilization: 0.009783

Average # of Tasks: 102.216305

Average Job Delay: 2997.775000

Shortest Processing Time

Average Finish Time: 2226.080000

Utilization: 0.019640

Average # of Tasks: 50.916743

Average Job Delay: -32.595000

Earliest Due Date

Average Finish Time: 2760.505000

Utilization: 0.017106

Average # of Tasks: 58.460504

Average Job Delay: 68.295000

Critical Ratios

Average Finish Time: 3005.690000

Utilization: 0.015153

Average # of Tasks: 65.993852

Average Job Delay: 76.860000

Slack per Remaining Operations

Average Finish Time: 2309.160000

Utilization: 0.019150

Average # of Tasks: 52.219810

Average Job Delay: 0.235000

Start conflicts, no due time conflicts:

First Come First Serve

Average Finish Time: 9241.110000

Utilization: 0.008659

Average # of Tasks: 115.485004

Average Job Delay: 514.280000

Shortest Processing Time

Average Finish Time: 2502.760000

Utilization: 0.015787

Average # of Tasks: 63.344976

Average Job Delay: -20.645000

Earliest Due Date

Average Finish Time: 2921.055000

Utilization: 0.014382

Average # of Tasks: 69.532373

Average Job Delay: 9.400000

Critical Ratio

Average Finish Time: 3717.370000

Utilization: 0.011199

Average # of Tasks: 89.295460

Average Job Delay: 23.540000

Slack per Remaining Operations

Average Finish Time: 2620.065000

Utilization: 0.015271

Average # of Tasks: 65.485254

Average Job Delay: 0.000000

Start conflicts, due time conflicts:

First Come First Serve

Average Finish Time: 8120.140000

Utilization: 0.011422

Average # of Tasks: 87.548679

Average Job Delay: 3464.515000

Shortest Processing Time

Average Finish Time: 1627.590000

Utilization: 0.028186

Average # of Tasks: 35.478801

Average Job Delay: -37.095000

Earliest Due Date

Average Finish Time: 2192.820000

Utilization: 0.022517

Average # of Tasks: 44.411544

Average Job Delay: 40.310000

Critical Ratio

Average Finish Time: 2703.030000

Utilization: 0.017649

Average # of Tasks: 56.661356

Average Job Delay: 181.920000

Slack per Remaining Operations

Average Finish Time: 1741.295000

Utilization: 0.026632

Average # of Tasks: 37.548140

Average Job Delay: 0.330000

VI. Conclusions & future work

The evaluation of the various schedulers discussed herein serves multiple purposes. First, it considers the utility of common scheduling approaches (familiar to most in the computer science domain) for satellite (and other cyber-physical systems with similar characteristics' scheduling). Second, through the concurrent evaluation of optimality and speed, it identifies characteristics that may make certain types of schedulers desirable for various applications, informing the creation of more adapted schedulers (based on the best aligned scheduling concept) for these applications. Finally, this work serves to inform the selection of a simple scheduler for use on CubeSats with limited computational capabilities.

Acknowledgment

Thanks is given to all of the students and faculty at UND that have participated in the design, development and testing of the OpenOrbiter spacecraft. Small satellite development work at the University of North Dakota is or has been supported by the North Dakota Space Grant Consortium, North Dakota NASA EPSCoR, the University of North Dakota Faculty Research Seed Money Committee, North Dakota EPSCoR (NSF Grant # EPS-814442), the Department of Computer Science, the John D. Odegard School of Aerospace Sciences and the National Aeronautics and Space Administration. The involvement of the numerous students and faculty from multiple disciplines in this project and the support of corporate sponsors (including GitHub, Inc.) is gratefully acknowledged.

References

[1] P. Dickson, *Sputnik: The Shock of the Century.* New York, NY: Walker Publishing Company, Inc., 2001.

[2] J. Cutler and G. Hutchins. OPAL: Smaller, simpler, and just plain luckier. 2000.

[3] R. A. Deepak and R. J. Twiggs. Thinking out of the box: Space science beyond the CubeSat. *Journal of Small Satellites 1(1),* pp. 3-7. 2012.

[4] J. Straub, "Cubesats: A low-cost, very high-return space technology," in *Proceedings of the 2012 Reinventing Space Conference,* Los Angeles, CA, 2012, .

[5] J. Garvey and E. Besnard. Development of a dedicated launch system for nanosat-class payloads. 2004.

[6] R. Milliron, "Interorbital's NEPTUNE dedicated SmallSat launcher:2013 test milestones and launch manifest update," in *2013 Spring CubeSat Developers' Workshop,* San Luis Obispo, CA, 2013, .

[7] G. Skrobot and R. Coelho. ELaNa–Educational launch of nanosatellite: Providing routine RideShare opportunities. Presented at Proc. SmallSat Conference. 2012, .

[8] M. Swartwout. AC 2011-1151: Significance of student-built spacecraft design programs it's impact on spacecraft engineering education over the last ten years. Presented at Proceedings of the American Society for Engineering Education Annual Conference. 2011, Available: http://www.asee.org/file_server/papers/attachment/file/0001/1307/paper-final.pdf.

[9] J. Straub and D. Whalen. Evaluation of the educational impact of participation time in a small spacecraft development program. *Education Sciences 4(1),* pp. 141-154. 2014.

[10] P. Thakker and G. Swenson. "Management and implementation of a cubesat interdisciplinary senior design course," in *Emergence of Pico- and Nanosatellites for Atmospheric Research and Technology Testing*, P. Thakker and W. Shiroma, Eds. 2010, .

[11] M. Swartwout. University-class satellites: From marginal utility to 'disruptive' research platforms. Presented at Proceedings of the 18th Annual AIAA/USU Conference on Small Satellites. 2004, .

[12] S. Padmanabhan, S. Brown, P. Kangaslahti, R. Cofield, D. Russell, R. Stachnik, J. Steinkraus and B. Lim. A 6U CubeSat constellation for atmospheric temperature and humidity sounding. 2013.

[13] I. Cartwright, L. Stepan and D. Lingard. Scheduling multi-spectral collection of the australian landmass using a 6U cubesat constellation. 2012.

[14] R. Staehle, D. Blaney, H. Hemmati, M. Lo, P. Mouroulis, P. Pingree, T. Wilson, J. Puig-Suari, A. Williams and B. Betts. Interplanetary CubeSats: Opening the solar system to a broad community at lower cost. Presented at CubeSat Developers' Workshop, Logan, UT. 2011, .

[15] D. Blaney, R. Staehle, B. Betts, L. Friedman, H. Hemmati, M. Lo, P. Mouroulis, P. Pingree, J. Puig-Sauri and T. Svitek. Interplanetary cubesats: Small, low cost missions beyond low earth orbit. Presented at Lunar and Planetary Institute Science Conference Abstracts. 2012, .

[16] B. Kading, J. Straub and R. Marsh. Mechanical design and analysis of a 1-U CubeSat. Presented at Presented at the North Dakota EPSCoR State Conference. 2014, .

[17] J. Straub, C. Korvald, A. Nervold, A. Mohammad, N. Root, N. Long and D. Torgerson, "OpenOrbiter: A Low-Cost, Educational Prototype CubeSat Mission Architecture," *Machines,* vol. 1, pp. 1-32, 2013.

[18] C. Korvald, J. Straub and H. Reza. Software architecture for a CubeSat that runs on open source software. *University of North Dakota Graduate School Scholarly Forum* 2013.

[19] D. Torgerson, J. Straub, A. Mohammad, C. Korvald and D. Limesand. An open-source scheduler for small satellites. Presented at SPIE Defense, Security, and Sensing. 2013, .

[20] J. Samson. Update on dependable multiprocessor CubeSat technology development. Presented at Aerospace Conference, 2012 IEEE. 2012, .

[21] C. Mitchell, J. Rexroat, S. A. Rawashdeh and J. Lumpp. Development of a modular command and data handling architecture for the KySat-2

CubeSat. Presented at Aerospace Conference, 2014 IEEE. 2014, . DOI: 10.1109/AERO.2014.6836355.

[22] C. Brandon and P. Chapin. "A SPARK/ada CubeSat control program," in *Reliable Software Technologies–Ada-Europe 2013*Anonymous 2013, .

[23] S. Fitzsimmons and A. Tsuda. Rapid development using tyvak's open source software model. 2013.

[24] S. Chien, J. Doubleday, K. Ortega, D. Tran, J. Bellardo, A. Williams, J. Piug-Suari, G. Crum and T. Flatley. Onboard autonomy and ground operations automation for the intelligent payload experiment (IPEX) CubeSat mission. 2012.

[25] J. Straub. A review of spacecraft AI control systems. Presented at Proc. 15th World Multi-Conference on Systemics, Cybernetics and Informatics. 2011, .

[26] D. W. Way, J. L. Davis and J. D. Shidner. Assessment of the mars science laboratory entry, descent, and landing simulation. 2013.