# Mars On-Site Shared Analytics Information and Computing

**Joshua Vander Hook,* Tiago Vaquero, Federico Rossi, Martina Troesch, Marc Sanchez Net, Joshua Schoolcraft, Jean-Pierre de la Croix, Steve Chien**
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

**Abstract**

We study the use of distributed computation in a representative multi-robot planetary exploration mission. We model a network of small rovers with access to computing resources from a static base station based on current design efforts and extrapolation from the Mars 2020 rover autonomy. The key algorithmic problem is simultaneous scheduling of computation, communication, and caching of data, as informed by an autonomous mission planner. We consider scheduling of a dependency chain of required and optional (but rewarding) tasks and present a consensus-backed scheduler for shared-world, distributed scheduling based on an Integer Linear Program. We validate the pipeline with simulation and field results. Our results are intended to provide a baseline comparison and motivating application domain for future research into network-aware decentralized scheduling and resource allocation.

## 1 Introduction

The NASA roadmap for 2020 and beyond includes several key technologies which will have a game-changing impact on planetary exploration. The first of these is High Performance Spaceflight Computing (HPSC), which will provide orders of magnitude increases in processing power for next-generation rovers and orbiters. In an effort to modernize the flight computing hardware available for NASA missions, the HPSC initiative was announced in 2013 (Doyle et al. 2013; Powell et al. 2011; Mounce et al. 2016; Schmidt et al. 2017). Unlike the current generation of computing, this program aims to keep NASA computing technologies at most one generation behind commercial technologies. HPSC is expected to become a mainstay in post-2020 deployments.

The second is Delay Tolerant Networks (DTNs), which overlay the Deep Space Network, providing internet-like abstractions and store-forward to route data through intermittent delays in connectivity. DTNs span communications links in an overlay architecture, enabling connectivity across network boundaries in a transparent manner, regardless of multiple potentially disparate network link layer protocols. A core principle of this overlay quality is the ability of individual nodes to store network data for possibly long durations before forwarding it to another node. Many features of Delay Tolerant Networking architectures are of particular utility in the deep space interplanetary communications realm, where a multitude of link layers, bandwidth constraints, and disruptions are expected during end-to-end transfer of mission commands and data (Wyatt et al. 2017; Cerf et al. 2007; Burleigh et al. 2003).

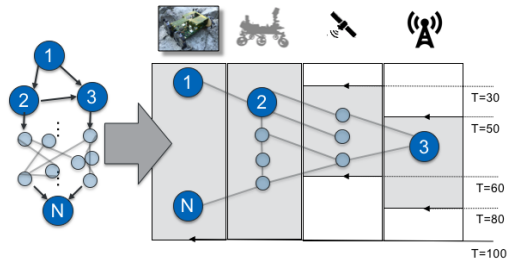



Figure 1: Illustrative MOSAIC scenario. A set of processing and data-driven tasks (left as dependency graph) must be mapped to multiple assets with heterogeneous computing, communication, and energy capacities. Each asset is also available over a fixed time window due to terrain effects or orbital parameters. The goal is to compute all the required tasks as quickly as possible.

The third is a trend toward small, co-dependent robots included in flagship missions (MarCO, PUFFER, and Mars Heli). Current planetary exploration is limited to benign operating areas due to the inability to land in or traverse challenging terrain. Unfortunately, the most compelling locations are often in these extreme terrains such as on Recurring Slope Lineae or in caves (McEwen et al. 2014; Léveillé and Datta 2010). In addition secondary mobile sensor platforms are beneficial because they can investigate transient targets without endangering the primary mission timeline e.g., being left behind to investigate the transient methane detection from MSL (Webster et al. 2015). Two examples of potential future systems being considered for development are the Mars Helicopter, and the "PUFFER" rover (Pop-Up Flat-Folding Explorer Robots) (Karras et al. 2017; Davydychev, Karras, and Carpenter 2019). Whatever the mission, these small craft can be released from parent rovers

*Corresponding Author hook@jpl.nasa.gov

or deployed in numbers via traditional landers and guided toward sampling targets. The "daughter-craft" often have even more restrictive weight, power, cost, and size constraints.

Combining these three trends, we envision scenarios in which a system containing two or more robotic agents with large discrepancies in processing power, communication bandwidths, data capacities, and energy storage must collaborate to achieve a science mission. We are motivated by the heterogeneity of these systems to study how the disparity in computation and communication capabilities affects small, resource-constrained, high-risk "edge" devices. In particular how, by optimizing data flows and processing assignments among all the devices, the mission efficiency can be increased. In this paper we formalize this problem and present preliminary results in modeling and analyzing Mars exploration missions. Because data and computation are shared among many devices, we dub a local computation-sharing network a MOSAIC (Multi-robot On-site Shared Analytics Information and Computing) network.

Our candidate mission involves one to fifteen daughter-craft deployed from a stationary lander to take and process samples from the environment. We present an Integer Linear Program that solves the optimal allocation of sensing, computation, and communication. We study the performance and show responsiveness to network changes in field demonstrations with multiple nodes. We demonstrate a 3x improvement in the number of samples taken by agents by redistributing the processing load to support agents in areas of interest. The tasks, communications models, and mission requirements are presented for reference and for comparisons by the community.

## 2 Related Work and Contribution

The problem of task scheduling at the core of the proposed MOSAIC architecture is known to be NP-complete (Garey and Johnson 1979; Ullman 1975). Furthermore, while polynomial-time approximation schemes for the problem exist, to the best of the authors' knowledge no such schemes are known for the task scheduling problem when computing nodes have *heterogeneous* computational capabilities (that is, the same task requires different computation times on different nodes) (Graham et al. 1979; Kwok and Ahmad 1999).

Accordingly, a large number of heuristic algorithms have been proposed to solve the task scheduling problem. Such heuristics may be classified as *list scheduling* heuristics (e.g. (Sih and Lee 1993)), which rely on greedily allocating tasks according to a heuristic priority task assignment; *clustering* heuristics (e.g. (Yang and Gerasoulis 1994)), which identify groups of tasks that should be scheduled on the same computing node; and *task duplication* heuristics, which duplicate some tasks to reduce communication overhead (e.g. (Ahmad and Kwok 1994)). In addition, a number of *guided random search* algorithms (including genetic algorithms (Yu and Buyya 2006) and ant colony optimization algorithms (Chen and Zhang 2009)) are available. We refer the reader to the survey in (Kwok and Ahmad 1999) and introduction in (Topcuoglu, Hariri, and Wu 2002) for a thorough review.

In particular, the heterogeneous earliest-finish-time (HEFT) heuristic algorithm (Topcuoglu, Hariri, and Wu 2002) provides excellent performance for heterogeneous task scheduling problems, and a number of variations of HEFT have been proposed (Tang, Li, and Padua 2009; Canon and Jeannot 2010; Tang et al. 2011). However, the HEFT algorithm and its derivatives generally assume that computation nodes are able to perform all-to-all communication and that the availability of communication links does not change with time; they also do not capture access contention or bandwidth constraints on communication links, and do not accommodate *optional* tasks which are not required to be scheduled but result in a reward when completed.

Several heuristics are also available for the *online* scheduling problem, where computational tasks appear according to a stochastic process, and are not revealed to the scheduler in advance (Tassiulas and Ephremides 1992; Dai and Lin 2005); recent work extends such schedulers to accommodate communication latency constraints (Yang, Avestimehr, and Pedarsani 2018). However, online approaches generally perform poorly compared to offline algorithms when the list of tasks to be executed is known in advance; in addition, even state-of-the-art online algorithms assume that all-to-all communication between the computation nodes is available. In contrast, the approach proposed in this paper (based on integer programming) does adapt to realistic communication constraints and accommodates optional tasks, while offering fast computation times that make the approach amenable for field use.

Specifically, our contribution is threefold. First, we design a scheduling algorithm based on integer programming that accounts for time-varying, bandwidth-constrained, multi-hop communication links and optional tasks and returns high-quality solutions in seconds. We also provide a distributed implementation of the algorithm based on a shared-world, consensus-backed model. Second, we validate the performance of the algorithm with hardware field tests. Third, we highlight a number of interesting emerging behaviors produced by the scheduler, and we show that sharing of computational tasks results in increased science throughput.

Collectively, the results in this paper show that sharing of computational tasks among heterogeneous agents is critical to realizing the benefits of heterogeneous multi-agent architectures, resulting in higher utilization of computational resources and increased scientific throughput for a given hardware architecture.

## 3 Problem Description

We now define the formulation of the communication and processing work flow and tasks that make up mission objectives. We consider robotic agents, each with an interdependent series of tasks, on-board computing, sensing, and a time-varying communication link to a subset of other agents.

**Agents:** Let there be $N \in \mathbb{Z}^+$ agents in the network, where $\mathbb{Z}^+$ denotes positive integers. The robot agents are denoted by $A_1, A_2, \ldots, A_N$. Each agent has known on-board processing, memory, and communication links.

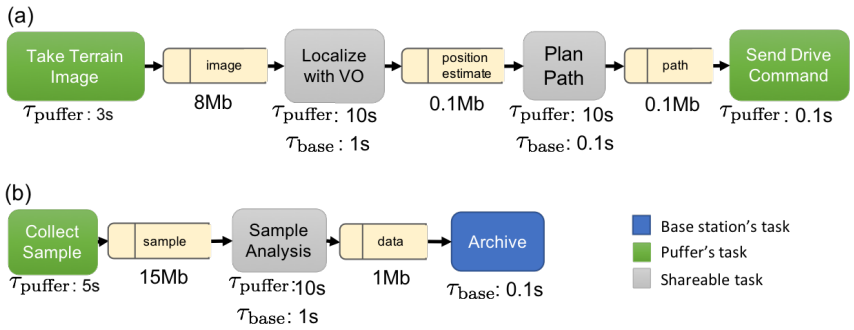



Figure 2: PUFFERs software network.

**Tasks and Software Network:** The agents perform $M \in \mathbb{Z}^+$ data-driven tasks. The set of $M$ tasks is denoted $\mathbb{T}$. A subset of the tasks are required, denoted $\mathbb{R} \subseteq \mathbb{T}$, meaning they must be executed each round. Optional tasks are things we would like to execute in addition to required tasks, and are given a reward score $r(T)$.

**Assumption:** An isolated agent can process all its own required tasks without communication or collaboration.

Tasks produce data products. Data products for task $T$ are denoted $d(T)$. The size of the data products are known a-priori as $s(T)$ for task $T$. Let $P_T$ be a set of predecessor tasks for $T$. So $j \in P_T$ means that task $T$ depends on the data product of task $j$. Data products from predecessors can be computed or communicated from other agents. We define the software network $SN$ as a dependency graph that encapsulates this information. A $SN$ used in our simulations is shown in Figure 2.

**Assumption:** Software networks $SN$ do not have cycles.

**Processors:** We consider heterogeneous processing times, so the time required to execute task $T$ on agent $i$ is given by: $\tau_i(T)$. The model represents, e.g., the worst-case, expected, or bounded computation time, and so all the times are deterministic. Program outputs are the same irrespective of the agent doing the computing (or are just as useful). If an agent has two or more *dissimilar* processing units, they can be modeled as coincident agents. If an agent has access to two or more *similar* processing units, we adjust the costs of each task to reflect its level of parallelization, but otherwise consider them the same processor.

A *solution* is a mapping of tasks to servers (agents) and start-times denoted $\mathbb{S} : i \rightarrow (A_j, t)$ where $j \in [1, \ldots, N]$ and $t \geq 0$. Each agent's computing schedule in a solution is denoted $\mathbb{S}_i = j \rightarrow_i (t)$.

**Communication Graph:** A key feature of DTN-based networking is Contact Graph Routing (CGR) (Wyatt et al. 2017; Araniti et al. 2015). CGR takes into account predictable link schedules and bandwidth limits to automate data delivery and optimize the use of network resources. The practical effect of incorporating DTN's store-forward mechanism into the scheduling problem is that it is possible to use mobile agents as *robotic routers* to ferry data packets past communication interference. The time-varying contact graph $CG$ captures the communication network topology between agents. For each agent, the graph provides a list of all the time intervals during which it can establish a directed communication link with another. An example time line representation for 3 agents with available bandwidths can be seen in Figure 3.

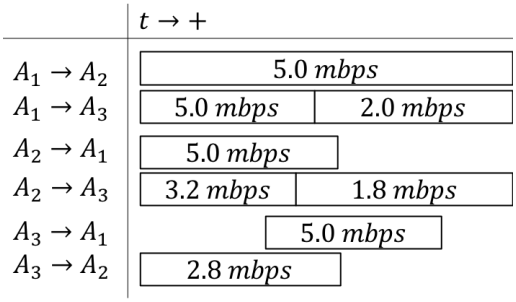



Figure 3: Contact graph for 3 agents showing connectivity time windows and bandwidths available.

Links have a time-varying data rate from 0 (not connected) to $\infty$ (communicating to self), denoted by $r_{ij}(t)$ for the rate from $A_i$ to $A_j$ at time $t$. At any time $k$, let $\mathcal{G}_k$ be the graph representing the set of agents it can send to or receive from, vertices $\mathcal{V} = \{1, \ldots, N\}$ and the directed edges $\mathcal{E}_k$ along which communication is possible. The task of communicating the data product $d(T)$ from $A_i$ to $A_j$ at time $t$ requires time $\tau_{ij}(T) \propto s(T)/r_{ij}(t)$ for *both* agents.

**Assumption:** Agents take 0 time to communicate the solution to themselves, except in the case of multiple onboard processors, which are modeled as coincident processors linked by a given data rate.

**Problem 1 (Distributed Computation)** *Given a set of tasks modeled as a software network $SN$, a list of computational agents $A_i$ $i \in [1 \ldots N]$, a contact graph $CG$, and a maximum schedule length $C^\star$, find a solution which is a mapping of tasks to servers (agents) and start times, $\mathbb{S} = f(i) :\rightarrow (A_j, t)$, such that: (1) The maximum overall computation time, $C(\mathbb{S}) = \max_j C(\mathbb{S}_j)$ is no more than $C^\star$; (2) All required tasks are scheduled; (3) All of the prerequisites for all required tasks are scheduled; (4) The reward due to optional tasks $\sum_{T \in \mathbb{T} \setminus \mathbb{R}} r(T) 1_{\{T \text{ is scheduled}\}}$ is maximized.*

## 4 Scheduler Implementation

To study the role of optimal distributed computing in our mission concepts, we cast Problem 1 as an integer linear program (ILP). We consider a discrete-time approximation of the problem with a time horizon of $C_d^\star$ time steps corresponding to the maximum schedule length $C^\star$. The optimization variables are: $X$, a set of Boolean variables of size $N \cdot M \cdot C_d^\star$. $X(i,T,c)$, with a true entry iff agent $A_i$ starts computing task $T$ at time $c$. $D$, a set of Boolean variables of size $N \cdot M \cdot C_d^\star$. $D(i,T,c)$, with a true entry iff agent $A_i$ has stored the data products $d(T)$ of task $T$ at time $c$. $C$, a set of Boolean variables of size $N^2 \cdot M \cdot C_d^\star$. $C(i,j,T,c)$, with a true entry iff agent $A_i$ communicates the product of task $T$ to agent $A_j$ at time $c$. The optimization objective is to maximize the sum of the rewards corresponding to completed optional tasks, i.e.

$$\sum_{i=1}^{N} \sum_{T \in \mathbb{R}} \sum_{c=1}^{C_d^\star - \tau_i(T)} r(T) X(i,T,c) \tag{1}$$

The problem constraints are captured by the following set of equations:

$$\sum_{i=1}^{N} \sum_{c=1}^{C_d^\star - \tau_i(T)} X(i,T,c) \leq 1 \quad \forall T \in \mathbb{R} \tag{2a}$$

$$\sum_{i=1}^{N} \sum_{c=1}^{C_d^\star - \tau_i(T)} X(i,T,c) = 1 \quad \forall T \in \mathbb{T} \setminus \mathbb{R} \tag{2b}$$

$$X(i,T,c) \leq D(i,L,c) \quad \forall i \in [1,...,N], T \in [1,...,M], L \in P_T, c \in [1,...,C_d^\star] \tag{2c}$$

$$\sum_{T=1}^{M} \left[ \sum_{j=1}^{N} / \left( C(i,j,T,c) + C(j,i,T,c) \right) + \sum_{\hat{c}=\max(1,c-\tau_i(T))}^{c} X(i,T,\hat{c}) \right] \leq 1 \quad \forall i \in [1,...,N], c \in [1,...,C_d^\star] \tag{2d}$$

$$D(i,T,c+1) - D(i,T,c) \leq \sum_{\tau=1}^{c} \sum_{j=1}^{N} \frac{r_{ji}(c)}{s(T)} C(j,i,T,c) + \sum_{\tau=1}^{c-\tau_i(T)} X(i,T,c) \quad \forall i \in [1,...,N], T \in [1,...,M], c \in [1,...,C_d^\star - 1] \tag{2e}$$

$$C(i,j,T,c) \leq D(i,T,c) \quad \forall i,j \in [1,...,N], T \in [1,...,M], c \in [1,...,T] \tag{2f}$$

$$D(i,T,1) = 0 \quad \forall i \in [1,...,N], T \in [1,...,M] \tag{2g}$$

Equation (2a) ensures that all required tasks are performed and (2b) that optional tasks are performed at most once. Multiple occurrences of the same task are modeled as separate tasks. Equation (2c) ensures that agents only start a task if they have access to the data products of its predecessor tasks from their own processing or from communication by one or more agents. Equation (2d) captures the agents' limited computation resources by ensuring that each agent either performs a single task, communicates, or is idle at any given time. Equation (2e) ensures that agents learn the content of a task's data products only if they (i) receive such information from other agents or (ii) complete the task themselves. Equation (2f) ensures that agent only communicate a data product if they have stored the data product themselves. Finally, Equation (2g) models the fact that data products are initially unknown to all agents.

The ILP has $N^2MC_d^\star + 2NMC_d^\star$ Boolean variables and $M(N(3C_d^\star - 1) + N) + NC_d^\star$ constraints; instances with dozens of agents and tasks and horizons of 50–100 time steps can be readily solved by state-of-the-art ILP solvers.

### 4.1 Distributed Implementation

In order to provide a *distributed* implementation of the scheduler presented above, we embedded it in a broadcast, plan, and execute cycle. The agents are assumed to have access to a common clock, and we consider an operational cycles 45 seconds long, where 5 seconds are dedicated to broadcasting the agents' states through a flooding mechanism, 10 seconds are dedicated to planning/scheduling, and 30 seconds are allocated to execution.

During the broadcast phase, agents learn the current state of the network, including set of tasks for all agents and communication link status. This has the added benefit of allowing online discovery of task lists each round, so the implementation is responsive to varying load and agent needs. However, the choice of a single broadcast epoch per round does cause some delay in responsiveness, since new tasks can only be scheduled if they appear prior to the broadcast cut-off for each round. A variety of gossip and consensus algorithms are available that provide better performance even in presence of time-varying communication links; the selection of a specific consensus algorithm is beyond the scope of this paper.

Then, all agents solve Problem 1with the network topology and vehicle states as inputs. Finally, each vehicle reads the scheduler's output and executes the tasks that are assigned to itself according to the prescribed timing.

The execution phase was chosen to be 30 seconds based on the autonomy tasks we modeled (Shown in Figure 2, discussed in Section 5 in particular Figure 4).

Problem 1 is in general is NP-hard, and a solver may fail to find an optimal solution within the allocated time. To ensure that a feasible final solution is found, we provide the solver with a trivial initial solution which consists of every agent performing only its own required tasks (the zero-reward case). The solver used is deterministic (i.e., it explores the decision tree according to a deterministic policy), and all agents terminate the solver after a prescribed, deterministic number of iterations, to ensure that each agent finds the same candidate solution when the timeout is reached.

The consistency of the scheduler output among all agents is controlled by using the same four inputs on all agents: i) All software networks from all agents ii) Status of all communication links iii) number of iterations allowed to the solver and iv) the initial conditions for the solver. Through this approach, we provide a *distributed* and *anytime* implementation of Problem 1.

We remark that the approach is not robust to failures of the broadcasting synchronization mechanism. The integration of more robust coordination mechanisms (e.g. challenge-response or watchdogs triggering the execution of an agreed-upon contingency plan) are an interesting direction for future research.

In the next section, we evaluate the performance of the proposed approach through software and hardware experiments.

## 5 Experimental Results

In this paper we focus on a Mars multi-robot science mission scenario based on current design efforts and extrapolation from the Mars 2020 rover autonomy. We study the decentralized task allocation and computation resource sharing in a varying network topology and bandwidth in which multiple smaller vehicles perform both maintenance tasks (e.g., sensing, path planning) and science tasks (e.g., microscope measurements) depending on where they are located. The goal is to maximize the number of science tasks executed, analyzed, and stored in a base station for further uplink.

We have implemented a hybrid framework that integrated simulated components and hardware implementations to test the distributed approach and the proposed scheduler. We explore different network topologies and present how the network of agents adapts to the changing environment.

### 5.1 Setup

Our scenario, illustrated in Figure 5a, is based on multiple PUFFERs cooperatively (i.e., their autonomous operations are coordinated by sharing information and tasks) exploring particular target science areas (blue zones) to support a parent platform (e.g., base station or flagship rover).

In practice, PUFFERs create a communication network during exploration and transit through the environment. The communication between vehicles varies depending on their distance and might be interrupted due to radio obstacles. We vary the number of PUFFERs from 3 to 15 to investigate the proposed scheduler's solutions as nodes come in and out of the network, the topology and bandwidth changes, and as they move around the environment.

Details of the current PUFFER hardware design are given in (Karras et al. 2017).

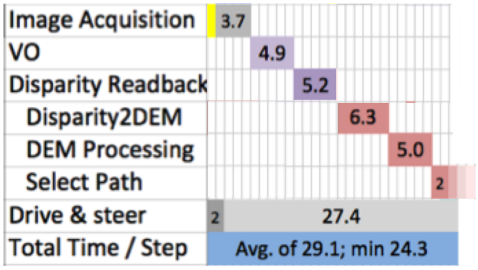



Figure 4: A simplified model for the Mars 2020 autonomous navigation pipeline from (Rieber 2017).

To design a representative software network, we borrowed from the current state of the art in the Mars 2020 rover (Rieber 2017). A highly simplified version of Mars 2020's autonomous navigation pipeline, including the timings for each task and waterfall dependencies on prior tasks, is shown in Figure 4. All tasks were previously run on representative hardware (RAD750), and their timings are shown in seconds.

Accordingly, each PUFFER, regardless of their position and zone, have to schedule a set of mandatory activities based on the task network illustrated in Figure 2 (top) which includes (i) capturing an image of the terrain, (ii) localizing itself based on that image, (iii) planning a path through the environment, (iv) and dispatching the drive command. While image capture and drive command have to be executed on board, localization and path planning tasks (both computationally more demanding) can be allocated to another vehicle in the network.

We assume the PUFFERs are exploring a distributed, but spatially-correlated phenomena, such as water moisture levels. We model the sampling and estimation on a similar terrestrial process used in farms (Tokekar et al. 2016). The point samples of moisture levels are gathered by spectroscopy or dipole measurements, and are incorporated into a spatial-estimation technique called Kriging (Bárdossy and Lehmann 1998). Kriging is computationally expensive, and requires storage of all measurements — not suitable for computationally-constrained devices like PUFFERs.

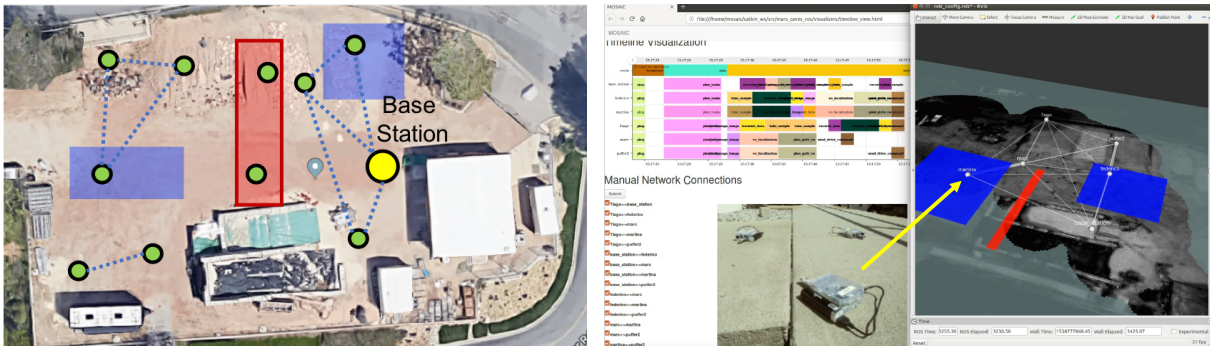
Accordingly, if a PUFFER is located in a science zone (blue) it can perform additional, optional, science tasks, as shown in Figure 2 (bottom). Specifically, a PUFFER can (i) take a sample from the environment, (ii) analyze it, and (iii) send the analysis data to the base station for storage and eventual uplink. The sample analysis task can be assigned to another vehicle, if appropriate, while storage can only be performed by the base station. Each one of these three tasks has a rewards associated to it; the reward for sample collection is set to 5, the reward for data analysis is 10, and the reward for storing data is 20.

When tasks are shareable and assigned to other vehicles, the scheduler needs to take in consideration the transfer times of the required data products. The duration of a transfer is determined by the bandwidth between the vehicles and the size of the data product (also shown in Figure 2). In our experiments, communication is blocked if the line-of-sight between two vehicles crosses the red zones. We also allow manually breaking a communication link to study the adaptation of the system. Herein, we used a step-wise function to model the communication bandwidth between nodes, from 1 Mbps within 15-200 meters up to 11 Mbps within 0-5 meters.

**Base Station** As a stationary resource, the base station includes more significant computational resources, such as an HPSC. The base station is also large enough to have more power generating capacity. To receive science data from the PUFFERs, the base station is equipped with the same communication equipment as the other nodes in the network.

The base station is not assigned any required tasks. Instead, it serves as a supporting node for sharing the computational load of the network, since it can perform all shareable tasks much faster compared to the PUFFERs (see cost comparisons in our Software Network in Figure 2). In addition, the base station is the only vehicle capable of storing the analyzed data in this scenario, acting as a notional cache for later downlink to Earth.

**Integrated Framework** Our framework integrates both 1) hardware and 2) software environment to simulated the components to control the vehicles tasks and communication. The PUFFERs were represented by Raspberry Pis (model




(a) Illustrative scenario in the Mars Yard at JPL.

(b) Visualization tools.

Figure 5: Experimental setup and RVIZ visualization. Timelines (left) represents the operational cycle and the task allocation. Communication links can be disabled to test system adaptation and relocation of tasks. RVIZ view (right) provides vehicle positioning and network topology information.

3) which were used to provide the location of the vehicle in the environment through a GPS antenna (see Figure 5b). The base station was represented by a desktop computer in the field. All the vehicles were connected through a WiFi router.

The main software environment that represents the logical layer of our experiment was implemented using the Robot Operating System (ROS). For each vehicle in the network we developed a set of simulated components, including a communication system to handle the broadcasting phase and transfer of data, a science component to emulate science tasks, a navigation/localization system to keep track of the vehicles pose, and a controller to plan and dispatch tasks according to the scheduler. In that environment a set of tools were developed to visualize 1) the execution timeline (and task allocation), 2) the network topology and bandwidth variations, 3) the vehicles positions in the environment, and 4) also to manipulate the scenario by adding additional no-communication zones, removing communication links. The software interface is shown in Figure 5b.

The ILP scheduler is designed to either run locally in each vehicle or as a service in the framework. The ILOG CPLEX solver was used to solve the ILP - since CPLEX does not support the Raspberry Pi's ARM architecture, we deployed the scheduler in an X86 server (AWS m5.xlarge) that was queried at each plan phase by each vehicle. Vehicles called the scheduler independently; the solution received by each vehicle was guaranteed to be consistent with the other vehicles' through use of a deterministic solver with a deterministic stopping criterion (discussed in Section 4.1) and caching.

The ILP solver's execution was terminated after a deterministic amount of solver steps (corresponding to approximately 10 seconds of execution on the Pi) to ensure consistency of the nodes' schedules, and the best solution found was returned. The solver was provided with an initial solution where agents did not share any computational tasks to ensure anytime availability of a feasible solution.

The performance of the system was evaluated in a field test. During the test, we added and removed PUFFERs from from the network (by activating and deactivating the corresponding Raspberry Pi's) in different locations; we moved active PUFFERs around, including in and out of science zones to change the set of tasks that must be scheduled; and we observed the solutions produced by the scheduler to study how tasks were allocated to (approximately) maximize science return. Representative portions of the field test are shown in a video in the Supplementary Material. In the next section, we present a set of interesting emerging task allocation scenarios from those observations.

## 5.2 Results

During field test, we observe a few common patterns in the task distribution among the network of agents while maximizing the number of science samples and their respective analysis and storage at the base. The presence of a communication barrier shown in Figures 5a and 5b was essential to create some of the interesting cases covered below.

**Science Clusters** One recurring behavior we observed refers to the formation of science clusters in which one vehicle is in the science region and one or more other vehicles are nearby but outside. A common situation in the experiment was to see a cluster of vehicle to the left of field in Figure 5a disconnected from another cluster to the right of the field, where the base station was located. In the left cluster, the vehicle in the science zone would off-load its localization and path planning tasks to the other nearby agent, so as to perform multiple sample collection and analysis tasks. *This represented a multiplicative increase in samples per round that can be collected and demonstrates the usefulness of edge distributed computation.* Due to the timing chosen for our tasks, the maximum increase achievable was 3x more samples per node in science zones. Occasionally, the data analysis task was also offloaded to nearby vehicles. Since the cluster on the left was disconnected from the base station, no data storage tasks were performed for samples collected from that cluster. Conversely, the cluster to the right of the field had the support of the base station, which allowed the PUFFERs in the science zone to not only col-

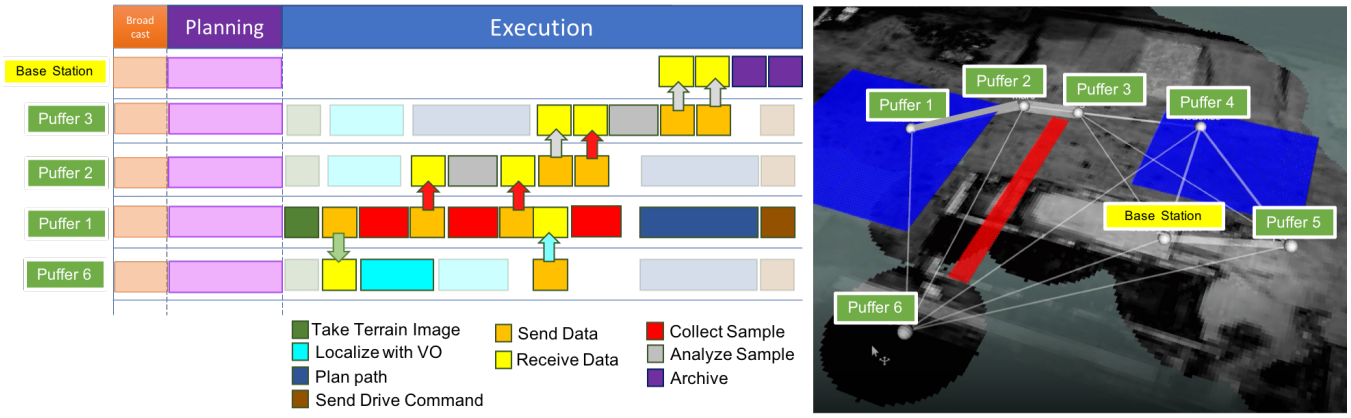



Figure 6: Illustrative example of the assembly line case.

lect more samples and off-load maintenance tasks, but also to store samples at the base station.

**Data Relay** Next, we strategically placed a few "bridge" nodes so that they would create communication links between nodes located in both of the aforementioned clusters. In this scenario, we observed some of the "bridge" nodes serving as data relays, transferring sample data or analyzed data from the left-hand-side cluster towards the base station. That relay behaviors was not limited to one node; several nodes could be used for that purpose, either as a chain of relays or just providing multiple options (and higher bandwidth) to transmit data from one cluster to the other.

**Science Assembly Line** As an extension of the previous scenario, Figure 6 (topology view on the right) shows three bridging nodes (PUFFERs 2, 3 and 6), in which some of the bridging vehicles (e.g, PUFFERs 2 and 3) would serve not only as data relays, transferring sample data, but also analyzing them on the way towards the base station. In additional to relay, some of the analysis task were also offloaded to those bridging node which created interesting cases of assembly lines. Figure 6 shows an illustrative example of that case, where PUFFER 1 is in the left-most science zone and offloads localization to the nearby PUFFER 6 (see green arrow in the timeline representing terrain image data being sent to PUFFER 6 which will later run a localization for PUFFER 6, cyan box). The figure shows the three sample collections in red by PUFFER 2. Only two sample data products are then transferred to a chain of bridging (in transit) nodes which themselves analyze the data and then transfer it to the base station for storage. In the example considered, PUFFER 2 receives two sample data products and send both analyzed data product and a raw sample data product to PUFFER 3, which exhibits a similar behavior and perform one analysis. The two analyzed data products are then forwarded (gray arrows) to the Base station. The third sample data product is not analyzed or stored given the lack of time for such; nevertheless, given that collecting a sample does add reward to the solution, the scheduler choose to do so.

This family of assembly line scenarios happened often when a chain of vehicles was established with a good bandwidth between the nodes, so that they could offload tasks more easily (especially to the base station) and transfer data at high rate. As mentioned, this demonstrates a 3x improvement in science samples that can be taken, simply by allowing edge distributed computation among agents, and orthogonal to any improvements in the network from e.g., autonomous path planning to move agents through the environment.

**Data Muling** An additional interesting *data muling* behavior was observed in numerical simulations (as shown in the video in the Supplementary Material). A cluster of three agents was connected to the base station via a single, low-bandwidth communication link. The agents were informed that one of the agents would drive towards the base station at the end of the optimization horizon; accordingly, agents knew that the bandwidth between the moving agent and the base station would increase, whereas the bandwidth between the same agent and the rest of the cluster would decrease. In this scenario, agents in the cluster transmitted data intended for the base station to the moving agent, effectively assigning it a "data mule" role; once the agent approached the base station, it relayed the relevant data to it. This scenario was not observed in field tests because, in those tests, the agents were manually moved, therefore the evolution of the link bandwidths was not predictable. Integration of the MOSAIC solver with the agents' autonomy stack, so as to predict future communication link availability based on planned movement, is an area for future research.

**Reproducing Our Results** To reproduce the scenarios we observed in field trials, or to reproduce on hardware, we have provided a public facing repository containing the core scheduler implementation and scenario descriptions that can be used as input to produce the discussed results. We provide a Python codebase compatible with the Robot Operating System (Quigley et al. 2009). In addition, a hardware implementation requires only a Raspberry PI and GPS module. See (Vander Hook 2019).

## 6 Conclusion

We described the MOSAIC concept for Mars exploration in which scheduling of computation, communication, and caching of data across networked assets is shown to be beneficial. We presented a series of scenarios to illustrate how MOSAIC networks can impact science utility, vehicle performance and enable an optimal distribution of computational loads, specially in multi-asset scenarios - a natural progression of future missions to Mars and other planets.

One defining feature of proposed Mars Sample Return mission concepts is the likelihood re-visiting the same area with subsequent launches to fetch, retrieve, and eventually launch soil samples for return to Earth (Mattingly and May 2011). If an on-site computing asset were available to multiple rovers in the area, they could make use of it for offloading their required engineering tasks, in order to take advantage of opportunistic science processing and sensing. Thus, the assisting asset(s) could provide an "infrastructure upgrade" and could remain on-site, providing communication, computation, and data analysis services for all subsequent phases of the campaign[1]. The asset could be embedded in a CubeSat network, and "piggy back" on the 2020 launch, similar to the MarCO CubeSats (Hodges et al. 2016; Schoolcraft, Klesh, and Werne 2016), be embedded in the "sky crane" lander and dropped during the "flyaway" phase (Korzun et al. 2010; Sell et al. 2013), could be a tethered balloon configuration (Kerzhanovich et al. 2004), or could be an aerostationary orbiter providing constant assistance to half the Mars surface(Breidenthal and Abraham 2016; Breidenthal et al. 2018). This concept is particularly compelling when assisting not just ground, but also orbiting networks. Our next efforts will address this domain.

The methods of this paper can be used to optimize the hardware of the distributed missions, or design communication networks for future Mars exploration missions, by simulating the scheduling problem in the loop with an iterative hardware trade explorer. Thus, determining the "tipping points" between different processing regimes is most important since the differences in efficiency between regimes can be very large. We expect this analysis will fold nicely into a framework similar to (Herzig et al. 2017) which provides a hardware-space expansion for designing multi-asset missions.

A primary next step is to investigate different scheduling techniques that could be utilized onboard the assets to allocate computation load. Agents might have different utility functions and goals that will add an interesting element to our network problem. Uncertainty and risk management is a key aspect of realistic assets networks for planetary exploration. Several aspects of exploration mission have uncertainty and can potentially be represented with stochastic models, such as task outcome and duration, vehicle failure, connectivity, bandwidth variations, and others. One promising research avenue is to incorporate probabilistic planning and scheduling approaches (Santana et al. 2016) to the computation sharing problem, as well risk-bounded techniques to provide guarantees that the network and the vehicles are able to operate within user specified bounds. Finally, different cost metrics merit study, in particular energy optimality, which is especially relevant given low-resource nodes like small ground robots.

[1] An interesting direction for future research would be to identify the requirements of such an asset.

## Acknowledgements

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

## References

Ahmad, I., and Kwok, Y.-K. 1994. A new approach to scheduling parallel programs using task duplication. In *Parallel Processing, 1994. ICPP 1994 Volume 2. International Conference on*, volume 2, 47–51. IEEE.

Araniti, G.; Bezirgiannidis, N.; Birrane, E.; Bisio, I.; Burleigh, S.; Caini, C.; Feldmann, M.; Marchese, M.; Segui, J.; and Suzuki, K. 2015. Contact graph routing in dtn space networks: overview, enhancements and performance. *IEEE Communications Magazine* 53(3):38–46.

Breidenthal, J., and Abraham, D. 2016. Design reference missions for deep-space optical communication. *The Interplanetary Network Progress Report* 42:205.

Breidenthal, J.; Xie, H.; Lau, C.-W.; and MacNeal, B. 2018. Space and earth terminal sizing for future mars missions. In *2018 SpaceOps Conference*, 2426.

Burleigh, S.; Hooke, A.; Torgerson, L.; Fall, K.; Cerf, V.; Durst, B.; Scott, K.; and Weiss, H. 2003. Delay-tolerant networking: an approach to interplanetary internet. *IEEE Communications Magazine* 41(6):128–136.

Bárdossy, A., and Lehmann, W. 1998. Spatial distribution of soil moisture in a small catchment. part 1: geostatistical analysis. *Journal of Hydrology* 206(1):1 – 15.

Canon, L., and Jeannot, E. 2010. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Transactions on Parallel and Distributed Systems* 21(4):532–546.

Cerf, V.; Burleigh, S.; Hooke, A.; Torgerson, L.; Durst, R.; Scott, K.; Fall, K.; and Weiss, H. 2007. Delay-tolerant networking architecture. Technical report.

Chen, W., and Zhang, J. 2009. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 39(1):29–43.

Dai, J. G., and Lin, W. 2005. Maximum pressure policies in stochastic processing networks. *Operations Research* 53(2):197–218.

Davydychev, I. A.; Karras, J. T.; and Carpenter, K. C. 2019. Design of a two-wheeled rover with sprawl ability and metal brush traction. *Journal of Mechanisms and Robotics* 1–12.

Doyle, R.; Some, R.; Powell, W.; Mounce, G.; Goforth, M.; Horan, S.; and Lowry, M. 2013. High performance spaceflight computing (hpsc) next-generation space processor (ngsp): a joint investment of nasa and afrl. In *Proceedings of the Workshop on Spacecraft Flight Software*.

Garey, M. R., and Johnson, D. S. 1979. *Computers and intractability*. W. H. Freeman.

Graham, R.; Lawler, E.; Lenstra, J.; and Kan, A. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In Hammer, P.; Johnson, E.; and Korte, B., eds., *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*. Elsevier. 287 – 326.

Herzig, S. J. I.; Mandutianu, S.; Kim, H.; Hernandez, S.; and Imken, T. 2017. Model-transformation-based computational design synthesis for mission architecture optimization. In *2017 IEEE Aerospace Conference*, 1–15.

Hodges, R. E.; Chahat, N. E.; Hoppe, D. J.; and Vacchione, J. D. 2016. The mars cube one deployable high gain antenna. In *2016 IEEE International Symposium on Antennas and Propagation (APSURSI)*, 1533–1534.

Karras, J. T.; Fuller, C. L.; Carpenter, K. C.; Buscicchio, A.; McKeeby, D.; Norman, C. J.; Parcheta, C. E.; Davydychev, I.; and Fearing, R. S. 2017. Pop-up mars rover with textile-enhanced rigid-flex pcb body. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 5459–5466. IEEE.

Kerzhanovich, V.; Cutts, J.; Cooper, H.; Hall, J.; McDonald, B.; Pauken, M.; White, C.; Yavrouian, A.; Castano, A.; Cathey, H.; Fairbrother, D.; Smith, I.; Shreves, C.; Lachenmeier, T.; Rainwater, E.; and Smith, M. 2004. Breakthrough in mars balloon technology. *Advances in Space Research* 33(10):1836 – 1841. The Next Generation in Scientific Ballooning.

Korzun, A. M.; Dubos, G. F.; Iwata, C. K.; Stahl, B. A.; and Quicksall, J. J. 2010. A concept for the entry, descent, and landing of high-mass payloads at mars. *Acta Astronautica* 66(7):1146 – 1159.

Kwok, Y.-K., and Ahmad, I. 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.* 31(4):406–471.

Léveillé, R. J., and Datta, S. 2010. Lava tubes and basaltic caves as astrobiological targets on earth and mars: a review. *Planetary and Space Science* 58(4):592–598.

Mattingly, R., and May, L. 2011. Mars sample return as a campaign. In *2011 Aerospace Conference*, 1–13.

McEwen, A. S.; Dundas, C. M.; Mattson, S. S.; Toigo, A. D.; Ojha, L.; Wray, J. J.; Chojnacki, M.; Byrne, S.; Murchie, S. L.; and Thomas, N. 2014. Recurring slope lineae in equatorial regions of mars. *Nature Geoscience* 7(1):53.

Mounce, G.; Lyke, J.; Horan, S.; Powell, W.; Doyle, R.; and Some, R. 2016. Chiplet based approach for heterogeneous processing and packaging architectures. In *2016 IEEE Aerospace Conference*, 1–12.

Powell, W.; Johnson, M.; Some, R.; Wilmot, J.; Gostelow, K.; Reeves, G.; and Doyle, R. 2011. Enabling future robotic missions with multicore processors. In *Infotech@ Aerospace 2011*. 1447.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 5. Kobe, Japan.

Rieber, R. R. 2017. Designing for a martian road trip: The mobility system for mars-2020. Keynote Talk: Mars Forum (URS: URS270204, CL17-5707).

Santana, P.; Vaquero, T.; Toledo, C.; Wang, A.; Fang, C.; and Williams, B. 2016. Paris: A polynomial-time, risk-sensitive scheduling algorithm for probabilistic simple temporal networks with uncertainty. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Schmidt, A. G.; Weisz, G.; French, M.; Flatley, T.; and Villalpando, C. Y. 2017. Spacecubex: A framework for evaluating hybrid multicore cpu/fpga/dsp architectures. In *Aerospace Conference, 2017 IEEE*, 1–10. IEEE.

Schoolcraft, J.; Klesh, A. T.; and Werne, T. 2016. Marco: interplanetary mission development on a cubesat scale. In *14th International Conference on Space Operations*, 2491.

Sell, S.; Chen, A.; Davis, J.; San Martin, M.; Serricchio, F.; and Singh, G. 2013. Powered flight design and reconstructed performance summary for the mars science laboratory mission. Technical report, Jet Propulsion Laboratory, National Aeronautics and Space Administration.

Sih, G. C., and Lee, E. A. 1993. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems* 4(2):175–187.

Tang, X.; Li, K.; Liao, G.; Fang, K.; and Wu, F. 2011. A stochastic scheduling algorithm for precedence constrained tasks on grid. *Future Generation Computer Systems* 27(8):1083 – 1091.

Tang, X.; Li, K.; and Padua, D. 2009. Communication contention in apn list scheduling algorithm. *Science in China Series F: Information Sciences* 52(1):59–69.

Tassiulas, L., and Ephremides, A. 1992. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE transactions on automatic control* 37(12):1936–1948.

Tokekar, P.; Hook, J. V.; Mulla, D.; and Isler, V. 2016. Sensor planning for a symbiotic uav and ugv system for precision agriculture. *IEEE Transactions on Robotics* 32(6):1498–1511.

Topcuoglu, H.; Hariri, S.; and Wu, M.-Y. 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 13(3):260–274.

Ullman, J. D. 1975. Np-complete scheduling problems. *Journal of Computer and System sciences* 10(3):384–393.

Vander Hook, J. 2019. NASA MOSAIC github repository. https://github.com/nasa/mosaic. [Accessed 2019-06-30].

Webster, C. R.; Mahaffy, P. R.; Atreya, S. K.; Flesch, G. J.; Mischna, M. A.; Meslin, P.-Y.; Farley, K. A.; Conrad, P. G.; Christensen, L. E.; Pavlov, A. A.; Martín-Torres, J.; Zorzano, M.-P.; McConnochie, T. H.; Owen, T.; Eigenbrode, J. L.; Glavin, D. P.; Steele, A.; Malespin, C. A.; Archer, P. D.; Sutter, B.; Coll, P.; Freissinet, C.; McKay, C. P.; Moores, J. E.; Schwenzer, S. P.; Bridges, J. C.; Navarro-Gonzalez, R.; Gellert, R.; and Lemmon, M. T. 2015. Mars methane detection and variability at gale crater. *Science* 347(6220):415–417.

Wyatt, E. J.; Belov, K.; Burleigh, S.; Castillo-Rogez, J.; Chien, S.; Clare, L.; and Lazio, J. 2017. New capabilities for deep space robotic exploration enabled by disruption tolerant networking. In *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 1–6.

Yang, T., and Gerasoulis, A. 1994. DSC: scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems* 5(9):951–967.

Yang, C.; Avestimehr, A. S.; and Pedarsani, R. 2018. Communication-aware scheduling of serial tasks for dispersed computing. In *2018 IEEE International Symposium on Information Theory (ISIT)*, 1226–1230.

Yu, J., and Buyya, R. 2006. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming* 14(3-4):217–230.