Noname manuscript No.
(will be inserted by the editor)

General queuing model for optimal seamless delivery of payload processing in multi-core processors

Esther Salamí · Cristina Barrado · Antonia Gallardo · Enric Pastor

Received: date / Accepted: date

Abstract Recent developments in Unmanned Aerial Systems (UAS) provide new opportunities in remote sensing applications. In contrast to satellite and conventional (manned) aerial tasks, UAS flights can be operated in a very short period of time. UAS can also be more specifically focused toward a given task such as crop reconnaissance or electric line tower inspection. For some applications, the delivery time of the remote sensing results is crucial. The current three-phase procedure of data acquisition, data downloading and data processing, performed sequentially in time, represents a drawback that reduces the benefits of using unmanned aerial systems. In this paper, we present a parallel processing strategy, based on queuing theory, in which the data processing phase is performed on board in parallel with data acquisition. The unmanned aerial system payload has been enlarged with low-cost, lightweight, multi-core boards to facilitate remote sensing data processing during flight. The storage of the raw sensing data is also done for possible further analysis; however, the ultimate decision support information can be seamless delivered to the customer upon landing. Furthermore, text alarms and limited imagery can also be provided during flight.

Keywords Multi-core · Queuing analysis · Real-time · Remote Sensing · Remotely Piloted Aircraft

Esther Salamí
E-mail: esalami@ac.upc.edu

Cristina Barrado
E-mail: cristina.barrado@upc.edu

Antonia Gallardo
E-mail: agallard@ac.upc.edu

Enric Pastor
E-mail: enric@ac.upc.edu

Department of Computer Architecture, Universitat Politècnica de Catalunya, Castelldefels, Spain.

1 Introduction

Unmanned aerial systems (UAS, also known as UAV, RPAS or drones) are increasingly being used to support remote sensing tasks [1–4]. In combination with or as a substitution for other remote sensing vehicles, UAS have the advantages of fast deployment and easy payload reconfigurability [5–7]. Moreover, UAS are well suited to poorly accessible areas and dangerous flight conditions [8–12]. In remote sensing tasks, their ability to operate very-low-level flights makes UAS highly useful. Nevertheless, most UAS used currently in remote sensing tasks acquire data during the flight and need later post-processing before the results are delivered. Assuming that flights are usually performed in remote areas, with no or slow communication facilities, the customer may have to wait one or two days before the final product is available.

For certain applications, two days of delay can be unacceptable. In precision agriculture, the watering levels, harvest time, and detection of plagues require fast responses. Some plagues are more vulnerable in a certain short period of time, during which they can be extinguished with a quick and cheap treatment. In these cases, a timely precision application reduces the impact of plagues and increases the harvest [13]. During a fire detection flight, notifications of hotspots need to be relayed as soon as possible. For homeland missions, the detection of and urgent alerting to migrant boats in real time can save lives. Currently, solutions to such time-restrictive applications are based on human operators. For instance, homeland drones fly beyond light of sight, use visual cameras and transmit video downstream using satellite communication [14]. A ground operator (who may also be the remote pilot) is devoted to the observation of the video and to raising an alarm when necessary. However, this human-based solution has two main drawbacks: the cost of the operation increases because of the high bandwidth communication and the operator costs, and the human factors on long, dull tasks can be concerning.

This paper explores a strategy for performing the necessary computation on-board the UAS during flight. For each specific mission, a particular sequence of data processing algorithms, mainly for image processing, needs to be continuously executed upon each new payload acquisition. That is, an ad-hoc pipeline of processing is constructed for each mission, therein conducting the flow of the sensed data. The execution time of the whole pipeline defines the minimum delay for the delivery of the final product. This time is on the order of a few seconds. However, given the continuous feed-in of sensing data into the processing pipeline, the throughput of the system at a given acquisition rate is limited by the slowest algorithm in the chain, which can block the complete flow.

Several inexpensive processing boards with multi-core capacity are currently available with commercial off-the-shelf technology. In this paper, queuing theory is used to pre-calculate the optimal pipeline configuration and maximum acquisition rate supported for each particular mission with the processing and acquisition resources available on board. Queuing models have being extensively used in many fields, especially in networking applications, resource optimization, and performance modeling of computer systems [15–20]. In their taxonomy of scheduling in general-purpose distributed computing systems [21], Casavant and Kuhl include queueing theoretic as a basic category of task allocation algorithms which can be used to arrive at an assignment of processes to processors. Chou and Abraham [22] use the general queueing model to derive closed form expressions to analyze the behavior of load redistribution algorithms. Deng and Purvis [23] develop a principle for multi-core task dispatching and validate the approach using exponential and deterministic models for packet processing and image search applications. Li [24] considers the problem of optimal partitioning of a multi-core server by modeling the server processor as a group of queueing systems.

Whereas in previous work [25] we relied on Markov $M/M/S$ queues, in this paper, we propose the use of general $G/G/S$ queues instead. The reason for this is that the arrival rate in these applications does not follow a random process; rather, it is fixed by the payload configuration and the mission requirements. On the other hand, the service time is given by the execution time of the data processing algorithms, whose probability distribution function is also far enough from an exponential distribution function. The number of servers has to be decided considering the limitations in terms of availability, weight, and power consumption. It is widely accepted that analytical queueing modeling is a cost-effective alternative to multi-core benchmark simulation in terms of both simulation time and resources. Nevertheless, the analysis of non-exponential queuing systems is mainly avoided because of analytical intractability reasons [22, 23]. In this study, the approximations presented in [30] are used for the general queue model analysis. This model provides performance metrics based on the first two moments of the general inter-arrival time and service-time distributions. Finally, an agent-based execution is used to validate the strategy on the two example missions.

The proposed queue-based strategy could also be applied to other conventional remote sensing platforms, such as manned aviation and satellites, if they possess on-board parallel computation capabilities with a flexible configuration. The problem is not specific of UAS or remote sensing tasks, but it differs from other scenarios like operating systems schedulers, where the inter-arrival time and processes to be executed are not known in advance.

The structure of this paper is as follows: Section 2 briefly describes the algorithms for payload processing and analyses their execution times on a multi-core board for several levels of parallelism. Section 3 presents the theory and the proposed strategy for applying the general queuing network model to schedule the parallel computation of the payload. In section 4, the strategy is validated using an agent-based execution that replicates the data flow of the flight using hardware in the loop. Finally, section 5 concludes the paper and provides future research directions.

Table 1: On-board data processing algorithms (VI = Visual, IR = Infrared)

Algorithm	Input	Output
Fusion	VI image, IR image, telemetry	TIFF image with thermal information on the visual image [27]
Georef	Telemetry, image pixel	Geographic coordinates using direct georeferencing [27]
Geotif	VI image, telemetry	TIFF image: undistort, georeference and rectify
Hotspot	IR image	List of hotspots: center of mass, bounding box, etc. [27]
Jellyfish	VI image	Image with bounding boxes and list of jellyfish [28]
Mosaic	4 × (VI image, telemetry)	TIFF image panorama using georeferencing
Overlap	2 × (VI image, telemetry)	Overlapping percentages of the two images
Quality	VI image	Blur (sharpness grade) and entropy (over- or under-exposure) metrics
Resize	VI image, new size	Scaled image
Stitch	4 × VI image	Panorama image using invariant local features [29]

2 Analysis of on-board data processing algorithms

A set of ten data processing algorithms have been developed by the research group for on-board processing the data captured by the payload. This section provides a general description of such algorithms together with an analysis of their execution on the oDroid-XU3 board [26]. It is assumed that this multi-core board is used as a co-processor, only dedicated to data processing tasks. The oDroid-XU3 is not involved in any flight management, guidance and control, conflict detection, or any other safety-critical tasks.

2.1 Algorithms description

The processing has been divided into stand-alone programs, and, depending on the UAS mission, several of the programs are executed in sequence. Table 1 lists the developed algorithms. The programs are written in C++ with the OpenCV3 [30] core library to support most image processing algorithms. Input data include images captured by visual and thermal cameras and telemetry, which is the position (latitude, longitude, and altitude) and attitude (yaw, pitch and roll) of the UAS. The resolution of the images used in this study is 320×240 pixels for thermal images and 5 MP for visual images. Output data can be either a geo-referenced or non-geo-referenced image, text with relevant information and/or one or more numeric values.

Table 2: Mean (τ) and coefficient of variation (c) of the execution time (in seconds) of data processing algorithms on the ODROID-XU3 for 1, 2, and 4 threads

	1 thread		2 threads		4 threads	
Algorithm	τ	c	τ	c	τ	c
Fusion	1.254	0.007	0.881	0.009	0.633	0.038
Georef	0.003	0.030	0.003	0.013	0.003	0.013
Geotif	1.125	0.008	0.744	0.008	0.542	0.010
Hotspot	0.036	0.064	0.034	0.069	0.034	0.068
Jellyfish	7.794	0.015	7.802	0.015	7.799	0.015
Mosaic	4.490	0.007	2.970	0.006	2.151	0.008
Overlap	0.777	0.070	0.477	0.063	0.316	0.059
Quality	0.555	0.023	0.514	0.026	0.497	0.027
Resize	0.336	0.014	0.286	0.017	0.261	0.024
Stitch	13.259	0.487	13.537	0.499	14.665	0.468

2.2 Execution time analysis

The execution time of the algorithms has been characterized on the ODROID-XU3 commercial off-the-shelf embedded board, which has two asymmetric quad-core CPUs (one Samsung Exynos5422 Cortex$^{\text{TM}}$-A15 2.0 GHz and one Cortex$^{\text{TM}}$-A7) [26]. The pthreads [31] parallelization framework already implemented in the OpenCV libraries was used to exploit the potential parallelism of the algorithms. The execution times were obtained from processing up to 100 images. The arithmetic mean (τ) and the coefficient of variation (c), which is the standard deviation of the execution time divided by its mean, are shown in Table 2. The results are given for execution with 1, 2, and 4 threads.

Notice that only four of the algorithms scale for parallel execution: *Fusion*, *Geotiff*, *Mosaic*, and *Overlap* (2.0X, 2.1X, 2.1X, and 2.5X performance speed-up, respectively, over 1 thread execution when they are running with 4 threads). Note also that most algorithms exhibit a coefficient of variation of approximately zero. This low variability is maintained when the number of threads increases. In general terms, we can say that, for the algorithms under study, the runtime follows a distribution that is much closer to a deterministic distribution (coefficient of variation equal to zero) rather than an exponential distribution (coefficient of variation equal to one). The exception is the *Stitch* algorithm, which exhibits the highest variability (coefficient of variation equal to 0.5). Having a number of cores available and with a low penalty in terms of power consumption for each additional core in use, the question becomes which is the best scheduling strategy for a given payload processing mission.

3 Resource optimization using the general queuing model

We consider a payload data processing application as a queuing model in which remote sensing data act as incoming clients that request a set of services. The offered service is the execution of a set of data processing algorithms. This section presents the use of the general queuing model, which considers the specific characteristics of the input rate and the execution time of on-board processing algorithms, as the optimizing strategy for on-board parallel execution of the payload data processing.

3.1 General queuing model

In Kendall's notation, the $G/G/S$ queue represents a system with S identical servers in parallel, unlimited queue length, and first-in first-out queue discipline, where inter-arrival times follow a general (arbitrary) distribution of average arrival rate λ and the service times follow a general independent distribution of average service rate μ (the inverse of the average service time τ) [32]. Particular cases are given for arrival rates and service rates that follow a Poisson distribution ($M/M/S$ model), also known as Markov processes [33], and for deterministic models, in which the inter-arrival time or service time is fixed and known ($D/M/S$ or $M/D/S$ models).

Metrics used to measure the performance of the queue include the average waiting time (W_q), average time in the system (W), average number of clients in the queue (L_q), average number of clients in the system (L), and utilization factor or traffic intensity (ρ). All these metrics are clearly related: first by the expressions $W = W_q + \tau$ and $L = L_q + L_s$, with L_s being the average number of clients in the servers; second by the definition $\rho = \lambda \cdot \tau / S$; and finally by Little's Law $L = \lambda \cdot W$ and $L_q = \lambda \cdot W_q$. Thus, the results herein will focus on W_q. A small W_q ensures that the waiting queues remain within tractable limits; however, a too small value will result in non-efficient resource usage. On the contrary, a large W_q represents a long waiting time, and when beyond the limit given by Eq. 1 (let us name it λ_{inf}), it makes the system non-stable.

$$\lambda < \lambda_{inf} = S \cdot \mu = S/\tau \tag{1}$$

An exact formulation can be used in certain models; however, approximations and/or computer simulations are required for more complex situations. In this study, the approximations presented in [34] are used for the $G/G/S$ queue analysis. This model depends on only five parameters: the arrival rate (λ), the squared coefficient of variation of the inter-arrival time (c_a^2), the average service time (τ), the squared coefficient of variation of the service time (c_s^2), and the number of servers (S).

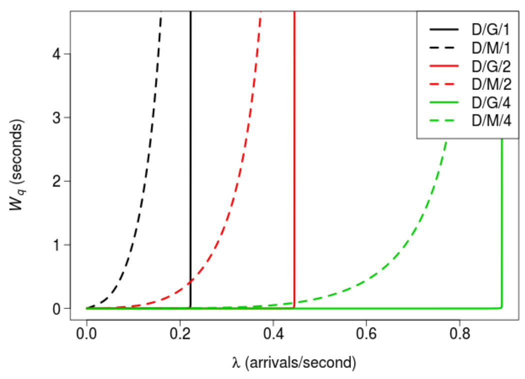
Most remote sensing applications use programmable cameras in which the acquisition rate can be set to a fixed value. This means that the time between consecutive input data is deterministic, in opposite to Poisson processes, which model random events. Furthermore, as seen in the previous section, the probability distribution function of the execution time of the algorithms for the payload data processing is also far enough from an exponential distribution function. Figure 1 shows the difference between the responses of both systems, the $D/M/S$ model (deterministic inter-arrival time and exponential service time, with mean service rate from Table 2) and the $D/G/S$ model (deterministic inter-arrival time and general service time, with mean and coefficient of variation from Table 2). The figure shows the system waiting time W_q as a function of the arrival rate λ. The results are given for two different image processing algorithms, *Mosaic* and *Stitch*, executing on 1 thread and for the number of servers S equal to 1, 2 and 4. The plots have been scaled to the average execution time of the algorithm (4.490 and 13.259 seconds, respectively). Solid lines represent the $D/G/S$ model and dashed lines represent the $D/M/S$ model.

Notice that both models satisfy Eq. 1 and λ asymptotically approaches λ_{inf} with increasing waiting time, even though the $D/M/S$ model has a smoother behavior in reaching the limit. For any given arrival rate lower than λ_{inf}, the average waiting time is always higher under the $D/M/S$ model compared to the $D/G/S$ model. This is perfectly reasonable because Markov queues model random behaviors. By comparing the $D/G/S$ plots of the two algorithms in Fig. 1, one can also notice the effect of the variability of the service time. While the *Mosaic* execution time has a very small coefficient of variation (0.007), the *Stitch* execution time exhibits higher variability (0.487). The effect of a small coefficient of variation is to provide a very pronounced change in the curve of the average waiting time. In contrast, a high coefficient of variation makes the curve closer to the $D/M/S$ curve.

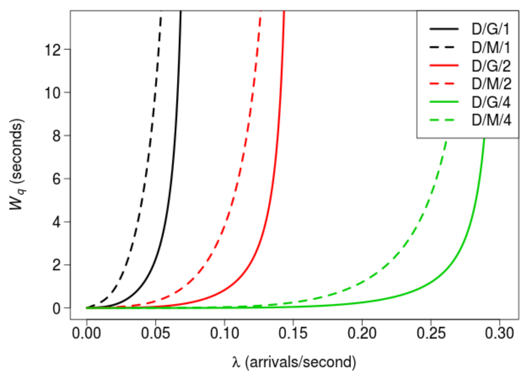
3.2 Heuristic for establishing Λ_{heu}

The goal of our payload processing architecture is to be able to process data at a reasonable arrival rate using a parallel hardware configuration that consumes less power and provides faster outputs. The arrival rate is limited by the speed of the sensor and by the maximum throughput of the system. For example, looking at the graphical response of the $D/G/S$ model in Fig. 1a, it can be inferred than the ODROID-XU3 is able to produce approximately 0.2 mosaics per second (one mosaic every 5 seconds) when using a single core. Considering that the *Mosaic* algorithm processes 4 input images to produce one output image, the system can manage a maximum camera capture rate of no more than 0.8 images per second (this is a minimum latency between consecutive images of 1.3 seconds) unless more resources are provided. When using the four cores to compute four consecutive mosaics in parallel, the maximum throughput increases up to approximately 0.9 mosaics per second (one image captured every 0.3 seconds).

Inequality in Eq. 1 provides a long-term stability condition of the system, but it is not enough to satisfy real-time constrains. In real-time systems, the



(a) Mosaic

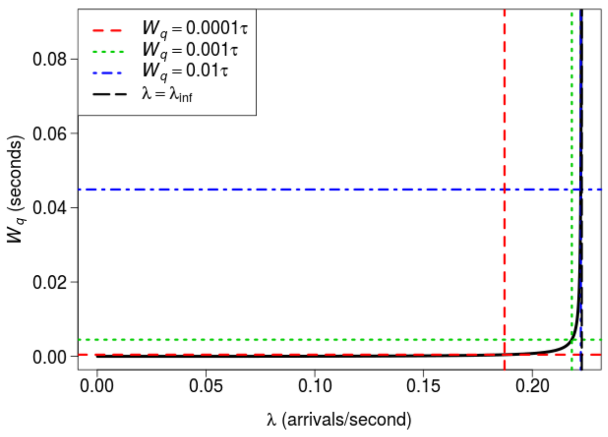


(b) Stitch

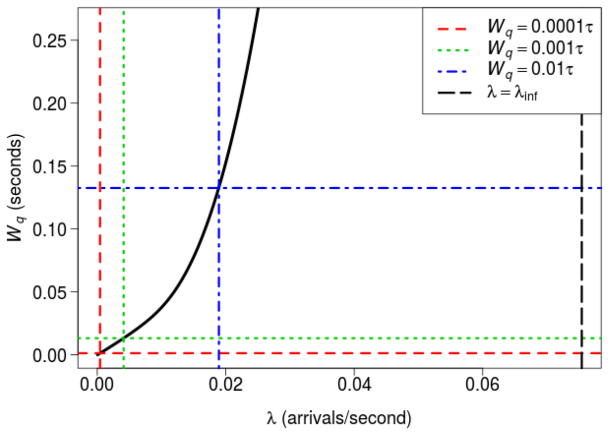
Fig. 1: Comparison of the average waiting time in the $D/G/S$ and $D/M/S$ models for the (a) *Mosaic* and (b) *Stitch* algorithms

time in which the actions take place is significant. Our goal is to find an optimal arrival rate Λ_{heu} that guarantees the highest possible flow, while trying to keep the average response time as short as possible. Being $W = W_q + \tau$, it entails to limit the maximum average waiting time W_q.

Figure 2 focuses on the response of the $D/G/S$ model when executing the *Mosaic* and *Stitch* algorithms on one core. Three thresholds are depicted: W_q equal to $1.00\%\,\tau$, $0.10\%\,\tau$, and $0.01\%\,\tau$. The points at which the threshold line cuts the plots sets the corresponding λ value. The closer the value from Λ_{heu} to Λ_{inf}, the closer we are to the optimal use of resources. But selecting Λ_{heu} equal to $1.00\%\,\tau$ leads to a waiting time is too close to the asymptote for a deterministic algorithm (see blue vertical line in Fig. 2a). To establish a balance between both extreme situations, we decided to use the next order of magnitude and limit the average waiting time up to $0.10\%\,\tau$. The value of



(a) Mosaic



(b) Stitch

Fig. 2: Selection of the upper threshold for the arrival rate in a service by limiting the waiting time up to 1.00% τ, 0.10% τ, or 0.01% τ for the (a) *Mosaic* and (b) *Stitch* algorithms

Λ_{heu} is then determined using Eq. 2.

$$\Lambda_{heu} = \{\lambda \mid W_q = 0.001\tau\} \tag{2}$$

The obtained Λ_{heu} for the *Mosaic* and *Stitch* algorithms is then 0.218 and 0.004 panoramas per second, respectively.

Table 3 shows the value of Λ_{heu} for the full set of data processing algorithms executing on 1 thread and for number of servers S equal to 1, 2 and 4. The Λ_{heu} value using the Markov model and the Λ_{inf} limit are also included for

Table 3: Λ_{heu} and Λ_{inf} of data processing algorithms executing on 1 thread on the ODROID-XU3 for the general and Markov models with 1, 2, and 4 servers

	1 server			2 servers			4 servers		
Algorithm	$D/M/1$ Λ_{heu}	$D/G/1$ Λ_{heu}	Λ_{inf}	$D/M/2$ Λ_{heu}	$D/G/2$ Λ_{heu}	Λ_{inf}	$D/M/4$ Λ_{heu}	$D/G/4$ Λ_{heu}	Λ_{inf}
Fusion	0.007	0.778	0.797	0.128	1.576	1.594	0.786	3.170	3.189
Georef	2.533	212.504	287.936	46.160	489.744	575.871	283.714	1057.171	1151.742
Geotif	0.008	0.865	0.889	0.143	1.754	1.778	0.876	3.531	3.556
Hotspot	0.247	13.567	28.038	4.495	36.781	56.076	27.627	88.143	112.152
Jellyfish	0.001	0.116	0.128	0.021	0.244	0.257	0.126	0.500	0.513
Mosaic	0.002	0.218	0.223	0.036	0.441	0.445	0.219	0.887	0.891
Overlap	0.011	0.589	1.287	0.206	1.625	2.574	1.268	3.955	5.147
Quality	0.016	1.468	1.802	0.289	3.235	3.605	1.776	6.815	7.210
Resize	0.026	2.718	2.977	0.477	5.686	5.954	2.933	11.629	11.908
Stitch	0.001	0.004	0.075	0.012	0.029	0.151	0.074	0.117	0.302

comparison. It can be seen that using the general model instead of the Markov model allows for more aggressive arrival rates.

3.3 Queuing network model

Up to this point, we have considered data processing algorithms as isolated services. As depicted in Fig. 3, the full mission can be modelled as a queue network in which remote sensing data act as incoming clients who request a sequence of such services (nodes). We consider the arrival rate to be deterministic for the first node of the network ($c_{a_1} = 0$) but general for the individual nodes following the first node.

In a steady-state, the average departure rate for each node should be equal to the average arrival rate to that node [35]. The coefficient of variation of the inter-arrival time to the node N_n (c_{a_n}) can be calculated using Eq. 3 [36].

$$c_{a_n}^2 = 1 + (1 - \rho_{n-1}^2)(c_{a_{n-1}}^2 - 1) + \frac{\rho_{n-1}^2}{\sqrt{S_{n-1}}}(c_{s_{n-1}}^2 - 1) \tag{3}$$

As expressed in Eq. 4, the throughput of the network is upper limited by two factors: first, by the maximum data acquisition rate (Λ_{acq}), which depends on the technical characteristics of the sensors being used; and second, by the maximum data processing rate supported by the network (Λ_{net}), which depends on the involved algorithms and available resources, and is limited by the lowest Λ_{heu} in the pipeline. Herein, we call $\Lambda_{mission}$ the upper threshold of the mission arrival rate.

$$\lambda_{out} \leq \Lambda_{mission} = \min\{\Lambda_{acq}, \Lambda_{net}\} \tag{4}$$

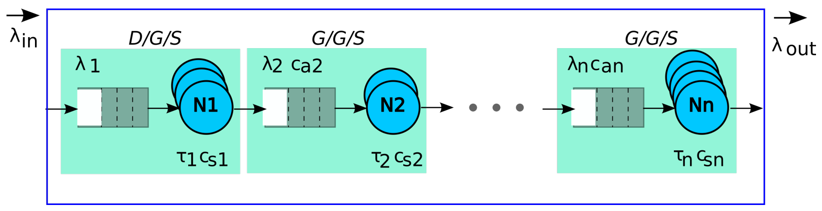


Fig. 3: Sequential network of queues. Arrival rate is deterministic for the first node of the network but general for the following nodes.

The average total service time of the nerwork is the sum of the individual service times, and the average waiting time is the sum of the average waiting times of the individual services.

3.4 Algorithm for setting the optimizing strategy and establishing Λ_{net}

As stated above, our goal is to establish an optimum configuration for the system to be able to process data arriving at a fixed arrival rate in real time and in the most efficient manner. The objective is minimizing the average time of the clients in the system by ensuring that the waiting queues remain within small limits. We propose an iterative method starting with the configuration that uses the minimum number of resources, that is, all algorithms running on a single core. Then, one additional resource is assigned on each iteration until either the desired arrival rate is reached or until there are no more resources (which means that it is not possible to work at the desired frequency with the available resources). Note that the maximum number of iterations is limited by the number of available resources. On each step, the mean and variability of the algorithm execution time is used to compute the upper threshold of the arrival rate in the queue network (Λ_{net}). If Λ_{net} does not fit the target arrival rate, then an additional core is given to the most restrictive algorithm, which is the algorithm with the lowest Λ_{heu}.

To illustrate the mechanism, we will focus on the case of a hotspot mission. In this mission, the UAS scans an area in a post-fire scenario in order to quickly detect hot areas and prevent fire revivals. The UAS payload consists of a thermal camera, a visual camera and a positioning system. The data processing is modeled as a queuing network that consists of the following sequence of algorithms: *Hotspots*, *Georef*, *Quality*, and *Fusion*. First, each thermal image is processed on-board with the *Hotspots* algorithm. If a hotspot is detected, the geographical position of the hotspot center of mass is calculated with *Georef*. In addition, the paired visual image is selected and processed: first, the *Quality* algorithm is executed, and upon achieving the positive threshold, the *Fusion* algorithm overlaps the thermal and visual images. For positive detections, both the information about the hotspot magnitude and geolocation and the fused image can be sent to the ground as a firefighters' alarm.

Table 4: Core allocation, τ_{net} (seconds) and Λ_{net} (arrivals/second) in the Hotspot mission (H=*Hotspot*, G=*Georef*, Q=*Quality*, F=*Fusion*)

	Algorithm	$S\times$threads	τ	Λ_{heu}	Core	τ_{net}	Λ_{net}
1	HGQF	1×1th	1.848	0.523	c_1	1.848	0.523
2	HGQ	1×1th	0.594	1.394	c_1	1.848	0.778
	F	1×1th	1.254	0.778	c_2		
3.a	**HGQ**	**1×1th**	0.594	1.394	$\mathbf{c_1}$	**1.475**	**1.092**
	F	**1×2th**	0.881	1.092	$\mathbf{c_2, c_3}$		
3.b	HGQ	1×1th	0.594	1.394	c_1	1.848	1.394
	F	2×1th	1.254	1.575	c_2, c_3		

From Table 3, it can be seen that *Fusion* is the most restrictive algorithm ($\Lambda_{heu} = 0.778$ images per second), whereas *Georef* is the least restrictive algorithm ($\Lambda_{heu} = 212.504$ images per second). Executing the four algorithms on one core (c_1) results in an equivalent service with a τ of 1.848 seconds per image and a Λ_{heu} of 0.523 images per second (iteration 1 in Table 4). The average service time of the equivalent service is computed as the sum of the average execution time of the algorithms; in addition, the coefficient of variation of the service rate has been calculated considering that the variance of the sum of the algorithms is the sum of their individual variances.

Imagine that we want to process one image per second ($\lambda_{in} = 1$), and suppose that this is supported by the maximum acquisition rate of the cameras (that is, $\Lambda_{acq} \geq 1$ image per second). Then, to ensure the proper functioning of the system, the pipeline configuration must provide $\Lambda_{net} \geq 1$ image per second. Because the initial Λ_{net} (0.523 images per second) does not satisfy this requirement, the most restrictive algorithm in the chain, that is, *Fusion*, is moved to a second core (c_2) (see iteration 2 in Table 4). In the first core, the service consisting of *Hotspots*, *Georef*, and *Quality* (HGQ) has a Λ_{heu} equal to 1.394 images per second, whereas the core executing Fusion (F) obtains the most restrictive Λ_{heu}, equal to 0.778 images per second. As a result, Λ_{net} is increased to 0.778 images per second, which is still below the desired limit. Because *Fusion* is again more restrictive than the other three algorithms together, an additional core (c_3) is assigned to it (iteration 3 in Table 4). Now, *Hotspots*, *Georef*, and *Quality* are executed on c_1, and *Fusion* is executed on c_2 and c_3. However, there are two options for utilizing the two cores: executing *Fusion* with 2 threads (3.a in Table 4) and executing two different instances of *Fusion* in parallel on each core (3.b in Table 4). Both options reach the target $\Lambda_{net} \geq 1$ image per second (Λ_{net} equal to 1.092 and 1.394, respectively). Thus, the option of choice would be 3.a because that option minimizes the execution time ($\tau_{net} = 1.475$ seconds) while ensuring the desired λ_{in}.

4 Model validation

Two use cases are run to validate the contributions of this paper. Each use case reproduces a UAS mission: a surveillance mission to detect jellyfish shoals, and a mission to detect of hotspots. The involved data processing algorithms are the following: *Jellyfish* for the first mission and *Hotspots*, *Georef*, *Quality* and *Fusion* for the second mission. For each mission, at least two runs are executed: one run with λ_{in} set to the $\Lambda_{mission}$ threshold obtained by the proposed algorithm and another run in which λ_{in} is set to $\Lambda_{mission} + 0.1$. Expectations are that the queue network is stable for λ_{in} equal to $\Lambda_{mission}$ and becomes unstable when a small increment of input flow is entered into the system. The data processing algorithms are executed on the actual UAS payload hardware (the ODROID-XU3) and process a sequence of one-hundred images taken from the visual and/or thermal cameras, with sizes of 5 M and/or 80 K pixels respectively. We assume that the sensors have a maximum Λ_{acq} equal to 1.0 image per second.

As in a real UAS flight, we build a parallel software system, in which each image processing algorithm is executed as an agent. Then UAS sensors are simulated with new agents that are programmed to publish an image at $\Lambda_{mission}$ rate. The communication between the agents is conducted via multi-language middleware, namely, the lightweight communication and marshaling (LCM) software bus [37], which supports the publish/subscribe communications paradigm. Figure 4 shows the example of the agents involved in the Hotspot mission. Observe that the agents involved in the data processing execute the real algorithms in the actual UAS hardware as if they were airborne. No physical allocation to the ODROID-XU3 cores is forced, and no specific priority is set to its Linux operating system scheduler.

Prior to the execution, the arrival rate of the mission is calculated using the resource allocation algorithm and the heuristic described in section 3.4. The algorithm returns a $\Lambda_{mission}$ threshold for λ_{in} and a specific configuration of the algorithms (sequential or parallel execution, one or multiple cores, and one or multiple threads). The involved agents are deployed on the hardware, with the specific configuration, while the sensor agents, which simulate the cameras, are deployed in another computer. The LCM middleware runs over the UDP protocol; thus, when an image is published but there is no service available, the image is simply lost.

4.1 Jellyfish: a use case of $D/G/S$ single-node queuing model to validate Λ_{heu}

Imagine a surveillance UAS with daily flights along a route parallel to the shoreline with the objective of informing the coast guards and the citizens about the proximity of jellyfish shoals. The UAS has a visual camera taking images of the water at regular intervals. The images are sent to the on-board processing system, which executes the jellyfish algorithm and obtains the number of jellyfish in the image. From the results, the UAS can send alarms in real

Table 5: Core allocation, τ_{net} (seconds) and Λ_{net} (arrivals/second) in the Jellyfish mission

	Algorithm	$S\times$threads	Core	τ_{net}	Λ_{net}
1	Jellyfish	1×1th	c_1	7.794	0.116
2	Jellyfish	2×1th	c_1, c_2	7.794	0.244
3	Jellyfish	3×1th	c_1, c_2, c_3	7.794	0.372
4	**Jellyfish**	**4×1th**	$\mathbf{c_1, c_2, c_3, c_4}$	**7.794**	**0.500**

time to the coast guards. In addition, the UAS can obtain relevant statistics about the jellyfish proliferation and their movements.

In the jellyfish mission, the network is composed of only one service, the *Jellyfish* algorithm. Thus, the Λ_{net} of this network will be equal to the Λ_{heu} of its single queue. The *Jellyfish* algorithm is CPU intensive (7.794 seconds). Because it is not scalable, the only reasonable parallelization strategy is to execute several jellyfish algorithms with a single thread, thus working in parallel on different images arriving at the queuing node.

We execute our iterative algorithm to find a faster but feasible input rate ($\Lambda_{mission}$) at which the system is not saturated. Observe in Table 5 the evolution of the iterations of the algorithm, starting with one core ($\Lambda_{net} = 0.116$ images per second still lower than Λ_{acq}), followed by 2 cores ($\Lambda_{net} = 0.244$ images per second), etc. The algorithm finishes after the 4th iteration (with $\Lambda_{net} = 0.5$ images per second) once all the resources (the 4 cores of the ODROID-XU3) have been assigned. From Eq. 4, we obtain that the $\Lambda_{mission}$ of the jellyfish mission is limited by Λ_{net}, this is, by the available computational resources. Thus, the acquisition rate of the camera ($\Lambda_{acq} = 1.0$ image per second) is not achievable for this mission.

We validate this result using the agent based software: 4 instances of the single-thread version of the *Jellyfish* algorithm are deployed, one instance for each processor core. When we set λ_{in} to 0.5 images per second, the $\Lambda_{mission}$ calculated threshold, the results show that all input images have been satisfactorily processed in slightly over than 3 minutes. In contrast, when λ_{in} is set to 0.6 images per second, a value above the λ_{inf} limit, the simulation ends faster; however, some processed images start failing after the 23rd image. From the total of 100 images, only 84 images were processed. The remaining 16 images were neglected because the agents were busy processing other images. Once the system starts to exhibit instability, the mission images start randomly missing at a rate of 2-3 images every 10 images. The execution shows that the proposed heuristic for selecting the Λ_{heu} of a service performs satisfactorily for a network consisting of a single node.

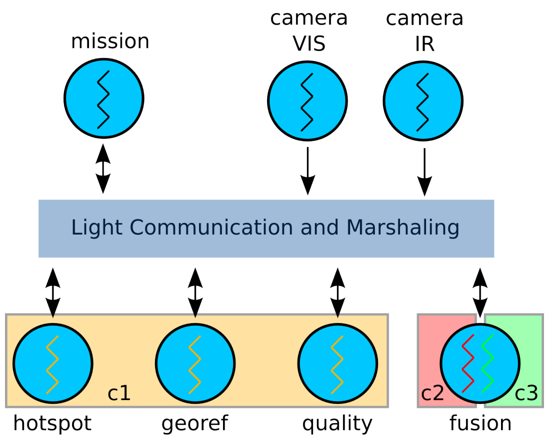


Fig. 4: Agent-based simulation setup

4.2 Hotspots: a use case of G/G/S queuing network model to validate Λ_{net}

The second use case is the hotspot mission presented in section 3.4. Figure 4 shows, below the LCM component, the 4 image processing agents of the hotspot mission, which constitute the queuing network. Above the LCM, we find the 2 agents simulating the cameras and the mission agent. The 2 cameras publish an image every λ_{in}^{-1} seconds. A total of 200 images (100 visual images and 100 thermal ones) are used for the validation. Approximately half of this set of images does not include a hotspot, whereas the other half does include a hotspot. This benchmark is set to simulate a flight wherein the initial surveillance is performed over a cold area, and when entering the hot area, almost all the images contain hotspots. The mission agent is responsible for the initial dispatching the parallel agents, acts as a central hub for all the mission messages, receives the results, and presents them to the end user. The specific configuration resulting from our algorithm is that the *Hotspots*, *Georef* and *Quality* algorithms are executed on one core, and the *Fusion* algorithm executes with 2 threads on another 2 cores. The 4th core remains inactive, thereby not consuming any power. In this case, $\Lambda_{mission}$ is limited by Λ_{acq}, this is, 1.0.

The execution of the validation when λ_{in} is 1.0 images per second creates a stable flow and finds all the hotspots in the last 49 images. The 49 tagged output images are correctly generated. In addition, the text information of each hotspot (its geolocation and magnitude) is returned. The list of hotspots may contain up to 9 hotspots, thereby being the most saturated input images processed consecutively in the simulation, in the same way that they would appear in a real flight.

Two more executions are also used to test situations with saturation. First, a λ_{in} of 1.1 images per second is attempted using the same parallel configuration and resource allocation as before. Then, a λ_{in} of 1.0 is also attempted but using the one-thread *Fusion* algorithm, therein executing the whole data processing network with only 2 cores. In the first case, the number of input images processed by the *Hotspots* algorithm is correct (100 images), and the number of geolocations returned is also correct (49 lists of hotspot locations); however, the number of fused images is only 32. For the second non-stable case, the algorithms executing on the first core perform correctly as before, but the *Fusion* algorithm is able to process only 25 images, namely, one every 2 images.

5 Conclusions

Real-time payload processing is a key feature that UAS should integrate on board to provide fast response to the end users of the system. Fire fighting and search-and-rescue tasks are clear examples of the usefulness of such immediate information. In addition, in precision agriculture, infrastructure maintenance, coastal guarding, etc., the rapid availability of results can be more useful than perfect accuracy. Quick response actions can be applied while being out in the field, rather than after returning to a computing facility, post-processing the payload data and providing results, which may already be obsolete after the processing period.

In this paper, we presented a method for matching the execution time of the payload processing necessities of a mission with available processing and acquisition capacities of on-board resources. Two contributions are presented: The first contribution is a heuristic for the selection of a suitable arrival rate of a service based on general queuing theory and applied to the real execution time and variability of the processing algorithms using the same multi-core board equipped by the UAS. The second contribution is the algorithm that selects the best resource allocation for a network of services composed of the processing algorithms executed in the pipeline, again using the extension of general queuing theory for networks of services. The proposed algorithm is fast and easy to implement. The algorithm obtains an optimal resource allocation of the payload services and an arrival rate that ensures the stability of the execution. The use of the general queuing model, instead of the previous Markov queuing model, results in higher arrival rates and thus a better utilization of resources.

Examples of payload processing are given using several image processing algorithms developed for different UAS missions. Ten independent algorithms, which use the OpenCV libraries compiled for parallel execution, are presented. Their execution times are given for an embedded, low-cost, low-power, multi-core board with sequential and parallel execution using 2-4 threads. Finally, a hardware-in-the-loop simulation is presented to validate the correctness of the contributions. The executions demonstrate the necessity of setting the correct parameters to ensure the stability of the system. Small variations in these parameters are also tested to demonstrate the negative effects that an incorrect estimation can produce.

In our mission implementation, with queues of length zero, any task arriving at the service was simply ignored if the resource was not available. In this sense, some of the images captured by the cameras are simply not processed. This may result in the necessity for a repeated flight if the number of missing images is excessive or if there is a missing output that is considered essential. With any luck, the flight could be repeated on the same day, thus only increasing flight costs and not field costs. Another solution to avoid the loss of images could be to store the pending tasks in a queue for later processing. However, if the network system is unstable, then the memory of the queues will overflow, and the results could be poor. For instance, a blocking of the processor, which could also be performing other critical tasks, or a fatal increase in the power consumption could affect the safety of the UAS.

Our immediate future work is to attempt the produce the presented solution for real UAS flights. An extension to additional alternative hardware boards and more missions also represents future work. New boards for integration include the 64-core Epiphany-IV and the small, low-cost and low-power Raspberry Pi. In parallel to this, efforts to improve the fine-grain parallelism shall be applied together with incorporation into our catalog of new payload processing algorithms required in future UAS applications.

Additional future work is the inclusion of more related parameters and functionalities as part of the queuing network. For instance, the UAS altitude and flight speed are directly related to the capturing setup of the camera. When requesting a mosaic of a flight area, for example, the images must overlap by 60-80 percent. Flying at high speeds may stress the requested acquisition rate; on the other hand, flying at high altitude can relax this requirement. In addition, the downstream communication of the results produced on board requires a limited bandwidth channel. The available bandwidth shall be in line with the throughput of the queuing network and with the image resolution of the equipment.

An important feature that requires further study is the power consumption. Especially for small UASs powered by batteries, the payload power consumption is a fundamental metric to be considered in the selection of the best strategy for achieving a successful mission. The inclusion of the power consumption can be addressed using a similar methodology based on a priori experimental profile generation; then, it can be modeled using a mathematical approximation function, which will then be incorporated into the network queuing model as part of the resources used in the system.

Acknowledgements This work has been partially funded by the Ministry of Economy and Competitiveness of Spain under grants number TRA2013-45119-R and TRA2016-77012-R.

References

1. Everaerts J (2009) NEWPLATFORMS - Unconventional platforms (Unmanned Aircraft Systems) for remote sensing. European Spatial Data Research (EuroSDR) Technical report 56 pp 58–103
2. Zhou G, Ambrosia V, Gasiewski AJ, Bland G (2009) Foreword to the Special Issue on Unmanned Airborne Vehicle (UAV) Sensing Systems for Earth Observations. IEEE Transactions on Geoscience and Remote Sensing 47(3):687–689
3. Colomina I, Molina P (2014) Unmanned Aerial Systems for photogrammetry and remote sensing: A review. ISPRS Journal of Photogrammetry and Remote Sensing 92:79–97
4. Salamí E, Barrado C, Pastor E (2014) UAV flight experiments applied to the remote sensing of vegetated areas. Remote Sensing 6(11):11,051, DOI 10.3390/rs61111051
5. Austin R (2010) Unmanned Aircraft Systems - UAVS Design, Development and Deployment
6. Watts AC, Ambrosia VG, Hinkley EA (2012) Unmanned Aircraft Systems in Remote Sensing and Scientific Research: Classification and Considerations of Use
7. Zhang C, Kovacs JM (2012) The application of small unmanned aerial systems for precision agriculture: A review. Precision Agriculture 13(6):693–712, DOI 10.1007/s11119-012-9274-5
8. Ackerman E (2011) Japan earthquake: Global Hawk UAV may be able to peek inside damaged reactors. Spectrum IEEE 17
9. Reavis B, Hem B (2011) Honeywell T-Hawk aids Fukushima Daiichi disaster recovery: Unmanned Micro Air Vehicle provides video feed to remote monitors. Honeywell Aerospace Media Center Honeywell International Inc 19
10. Baker RE (2012) Combining micro technologies and unmanned systems to support public safety and homeland security. Civil Engineering and Architecture 6(10):1399–1404
11. Turner D, Lucieer A, Watson C (2012) An automated technique for generating georectified mosaics from ultra-high resolution unmanned aerial vehicle (UAV) imagery, based on structure from motion (SfM) point clouds. Remote Sensing 4(5):1392–1410
12. Ambrosia V, Buechel S, Wegener S, Sullivan D, Enomoto F, Hinkley E, Zajkowski T (2011) Unmanned airborne systems supporting disaster observations: Near-real-time data needs. In: Proceedings of 34th International Symposium on Remote Sensing of Environment. CD Proceedings, paper reference, vol 144, pp 1–4
13. Oliveira I, Pereira JA, Lino-Neto T, Bento A, Baptista P (2012) Fungal diversity associated to the olive moth, Prays oleae Bernard: A survey for potential entomopathogenic fungi. Microbial ecology 63(4):964–974
14. Skinnemoen H (2014) UAV & satellite communications live mission-critical visual data. In: Aerospace Electronics and Remote Sensing Technology

(ICARES), 2014 IEEE International Conference on, pp 12–19, DOI 10.1109/ICARES.2014.7024391
15. Govil MK, Fu MC (1999) Queueing theory in manufacturing: A survey. Journal of manufacturing systems 18(3):214–240
16. Hsu CF, Liu TL, Huang NF (2002) Performance analysis of deflection routing in optical burst-switched networks. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol 1, pp 66–73, DOI 10.1109/INFCOM.2002.1019247
17. Menasce DA, Dowdy LW, Almeida VAF (2004) Performance by design: Computer capacity planning by example. Prentice Hall PTR, Upper Saddle River, NJ, USA
18. Qiu T, Feng L, Jiang H, Sun W (2013) Queueing model analysis and scheduling strategy for embedded multi-core SoC based on task priority. Computers & Electrical Engineering 39(1):24–33, DOI http://dx.doi.org/10.1016/j.compeleceng.2012.03.001, special issue on Recent Advanced Technologies and Theories for Grid and Cloud Computing and Bio-engineering
19. Munir A, Gordon-Ross A, Ranka S, Koushanfar F (2014) A queueing theoretic approach for performance evaluation of low-power multi-core embedded systems. J Parallel Distrib Comput 74(1):1872–1890, DOI 10.1016/j.jpdc.2013.07.003
20. Qiu T, Zhao A, Ma R, Chang V, Liu F, Fu Z (2016) A task-efficient sink node based on embedded multi-core soc for internet of things. Future Generation Computer Systems DOI http://dx.doi.org/10.1016/j.future.2016.12.024
21. Casavant TL, Kuhl JG (1988) A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Transactions on Software Engineering 14(2):141–154
22. Chou TCK, Abraham JA (1983) Load redistribution under failure in distributed systems. IEEE Transactions on Computers 32(9):799–808
23. Deng JD, Purvis MK (2011) Multi-core application performance optimization using a constrained tandem queueing model. Journal of Network and Computer Applications 34(6):1990–1996, DOI http://dx.doi.org/10.1016/j.jnca.2011.07.004, control and Optimization over Wireless Networks
24. Li K (2015) Optimal partitioning of a multicore server processor. The Journal of Supercomputing 71(10):3744–3769
25. Salamí E, Soler JA, Cuadrado R, Barrado C, Pastor E (2015) Virtualizing supercomputation on-board UAS. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XL-7/W3:1291–1298, DOI 10.5194/isprsarchives-XL-7-W3-1291-2015
26. Lee J (2014) ODROID-XU3: The fastest computer made by hardkernel so far! ODROID Magazine pp 22–23
27. Salamí E, Barrado C, Pastor E, Royo P, Santamaria E (2013) Real-time data processing for the airborne detection of hot spots. Journal of

Aerospace Information Systems 10(10):444–451

28. Barrado C, Fuentes Ja, Salamí E, Royo P, Olariaga aD, López J, Fuentes VL, Gili JM, Pastor E (2014) Jellyfish monitoring on coastlines using remote piloted aircraft. IOP Conference Series: Earth and Environmental Science 17:012,195, DOI 10.1088/1755-1315/17/1/012195
29. Brown M, Lowe DG (2007) Automatic panoramic image stitching using invariant features. International Journal of Computer Vision 74(1):59–73, DOI 10.1007/s11263-006-0002-3
30. Pulli K, Baksheev A, Kornyakov K, Eruhimov V (2012) Real-time computer vision with OpenCV. Communications of the ACM 55(6):61–69, DOI 10.1145/2184319.2184337
31. Lewis B, Berg DJ (1998) Multithreaded programming with pthreads. Prentice-Hall, Inc., Upper Saddle River, NJ, USA
32. Kendall DG (1953) Stochastic processes occurring in the theory of queues and their analysis by the method of the embedded markov chain. The Annals of Mathematical Statistics 24(3):338–354
33. Gautam N (2012) Analysis of queues: Methods and applications. CRC Press
34. Whitt W (1993) Approximations for the $GI/G/m$ queue. Production and Operations Management 2(2):114–161
35. Bertsekas D, Gallager R (1992) Data networks (2Nd Ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA
36. Whitt W (1983) The queueing network analyzer. Bell System Technical Journal 62(9):2779–2815, DOI 10.1002/j.1538-7305.1983.tb03204.x
37. Huang AS, Olson E, Moore DC (2010) LCM: Lightweight Communications and Marshalling. In: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pp 4057–4062, DOI 10.1109/IROS.2010.5649358