A Robust, Reliable and Energy-aware Urgent Computing Framework for Ensembles of Forecasts

Siew Hoon Leong

München 2016

A Robust, Reliable and Energy-aware Urgent Computing Framework for Ensembles of Forecasts

Siew Hoon Leong

Dissertation
an der Informatik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Siew Hoon Leong

München, den Abgabedatum

First Reviewer (Erstgutachter): Prof. Dr. Dieter Kranzlmüller
Second Reviewer (Zweitgutachter): Prof. Dr. Hans-Joachim Bungartz
Tag der mündlichen Prüfung: 13.06.2016

Contents

List of Figures

List of Tables

Abstract

Events like disasters can cause major economic losses and casualties and pose a challenge to political leaders to take appropriate prevention and mitigation actions. Forecasts of time-critical events such as flash floods and tsunamis must be completed before a stipulated deadline. The simulated results are required by relevant authorities to make timely educated decisions to mitigate the economic losses and reduce casualties. Due to the inherent uncertainties in most forecast models, stochastic methods based on an ensemble of forecasts are prevalent. Urgent computing intends to enable multiple forecasts with varying execution time to complete within a deadline. Consequently, a substantial number of different resources and an adaptable framework that can swiftly and effectively manage and allocate multiple resources for such computations is required.

This dissertation presents a comprehensive definition of urgent computing, a generic framework for managing its processes, and a set of robust, reliable and energy-aware resource allocation heuristics. The original definition of urgent computing has been evaluated in depth and extended to formalise its unique characteristics and requirements that were not clearly described in the original definition. A generic framework, consisting of four software component urgent managers, information, resource & environment, schedule and fault, is correspondingly designed to guide the fulfilment of these characteristics and requirements. A set of robust, reliable and energy-aware resource allocation heuristics integrated within the framework ensures that the most crucial requirement, the deadline, is met. Two models, robustness and reliability, are defined to quantify the heuristics. Energy-awareness is included to manage the energy constraints of resource providers when frequency scaling is adopted to improve total time required for the urgent computations. An assessment model is also introduced to evaluate and compare the allocation results among the heuristics.

The framework is verified with a real flash flood ensemble forecast where the required environment is prepared in advance. The verification environment includes a mix of real production and hypothetical high performance computing resources. The use of the hypothetical resources serves to increase the stress on the heuristics by including selected controlled variables into the experiment. Potential future work includes the addition of another criterion, computation cost, to the resource allocation heuristics, prediction models to evaluate the results of the computations and/or the use of advanced 3D visualisation to assist the decision making and coordination of mitigation activities.

Zusammenfassung

Naturkatastrophen und ähnliche Ereignisse können zu großen ökonomischen Verlusten und vielen Toten führen, und stellen politische Verantwortungsträger vor die Herausforderung, angemessene vorbeugende und lindernde Maßnahmen zu treffen. Vorhersagen von zeitkritischen Ereignissen wie z.B. Überschwemmungen oder Tsunamis müssen bis zu einer vorgegebenen Frist abgeschlossen sein. Die zustndigen Behörden benötigen Simulationsergebnisse, um rechtzeitig sachgerechte Entscheidungen zur Schadensminimierung treffen zu können. Aufgrund der inhärenten Unsicherheiten in den meisten Vorhersagemodellen werden oft stochastische Methoden verwendet, die auf einem Ensemble an Vorhersagen basieren. Mithilfe des Urgent Computing sollen mehrere Vorhersagen mit unterschiedlichen Laufzeiten bis zu einer vorgegebenen Frist ermöglicht werden. Daher braucht es eine Vielzahl an unterschiedlichen Ressourcen sowie ein adaptives Framework, das diese Ressourcen schnell und effektiv für solche Berechnungen zur Verfügung stellen und verwalten kann.

Diese Dissertation beginnt mit einer umfassenden Definition des Urgent Computing, und präsentiert dann ein generisches Framework zur Verwaltung seiner Prozesse, sowie mehrere robuste, verlässliche und energiebewusste Heuristiken zur Ressourcenzuteilung. Die ursprüngliche Definition des Urgent Computing wird im Detail untersucht und erweitert, um ihre spezifischen Charakteristiken und Anforderungen zu formalisieren, die in der ursprnglichen Definition nicht klar beschrieben wurden. Dementsprechend wird ein generisches Framework entworfen, das aus vier Software-Komponenten besteht (Urgent Managers, Information, Ressourcen und Umwelt, Scheduling und Strung), um diese Charakteristiken und Anforderungen zu erfüllen. Die Integration einer Reihe von robusten, zuverlässigen und energiebewussten Heuristiken zur Ressourcenzuteilung in das Framework stellt sicher, dass die Deadline als wichtigste Rahmenbedingung eingehalten wird. Es werden zwei Modelle definiert, Robustheit und Zuverlässigkeit, um die die Heuristik zu quantifizieren. Energiebewusstsein wird miteinbezogen, um die energetischen Rahmenbedingungen der Ressourcenanbieter zu erfüllen, wenn Frequenzskalierung verwendet wird, um die Gesamtlaufzeit der Urgent Computing-Rechnungen zu verringern. Schließlich wird ein Analyse- und Bewertungsmodell eingeführt, um die Zuteilungsergebnisse der verschiedenen Heuristiken einschtzen und zu vergleichen zu können.

Das Framework wird mithilfe eines realen Überschwemmungsvorhersage-Ensembles verifiziert, in dem die nötige Umgebung im Voraus präpariert wurde. Die Verifikationsumgebung besteht aus einer Mischung aus echten Produktionssystemen und hypothetischen High

Performance Computing-Ressourcen. Die hypothetischen Ressourcen werden hinzugezogen, um die Belastung der Heuristiken zu erhöhen, was durch Einbeziehen ausgewählter Kontrollvariablen in das Experiment erreicht wird. In möglichen Folgearbeiten knnte als zusätzliches Kriterium der Rechenaufwand in die Zuteilungsheuristik miteinbezogen werden, es könnten Vorhersagemodelle zur Auswertung der Berechnungsergebnisse erstellt werden, oder es knnten hochentwickelte 3D-Visualisierungstechniken verwendet werden, um Entscheidungsprozesse sowie die Koordination der Schadensbegrenzungsanstrengungen zu unterstützen.

Chapter 1

Introduction

Disaster mitigation [3] is critical to individuals and nations in the world. When disasters like earthquakes, tsunamis, forest fires, storms, floods, radiological accident, epidemic outbreaks, etc., are expected or have taken place, it is of utmost importance to make timely decisions for managing the affected areas and reduce casualties and economic losses. Numerical forecasts can generate information and provide predictions to facilitate this decision-making process. If the forecasts have to be initiated and computed within a (short) required timeframe as input data for the computations are only available shortly before/when/after the event strikes, this class of computing is referred to as urgent computing. Urgent computing enables responsible authorities to make educated decisions by providing simulated predictions/forecasts of disasters, the impact and required evacuation zones, etc., before a stipulated deadline. The applications of urgent computing are not limited to disaster scenarios and are relevant in a wider context, e.g. medical applications for in-surgery support and financial applications for crisis analysis, where imminent losses are expected. In this dissertation, we will focus on disaster scenarios.

1.1 Motivation and Research Question

Disasters often occur unexpectedly anywhere and anytime, and frequently have devastating impact economically and can result in loss of human lives. Based on the data collected by EM-DAT's International Disaster Database[1] shown in Figure 1.1, there is an increasing trend in the number of reported natural disasters and economic damage.

Figure 1.1a illustrates the total number of disasters per continent (y-axis) from the year 1900 to 2015 (x-axis). No data is available before 1900. The Asia continent has the most significant increase in the number of disasters. The Africa, Americas and Europe

[1] http://www.emdat.be

continents have also seen a marked increase while the Oceania continent faces a gradual increase. The green line with square markers shows the number of disasters across all continents, revealing an exponential increase in natural disaster occurrences over the last century.

Figure 1.1b depicts the total economic damage in USD (billion) (y-axis) from the year 1900 to 2015 (x-axis) for different natural disaster types. The most damaging disasters, earthquakes, floods, storms, droughts and epidemics are highlighted as separate plots. Other disaster types (yellow line with star markers) include animal accidents, insect infestations, wildfires, extraterrestrial impacts, mass movement (dry), volcanic activities, landslides and extreme temperatures. All six plots exhibit exponential increases in the total economic damage over the years.

The 2015 annual Global Assessment Report [4] from the United Nation office further shows that economic losses from disasters are in fact reaching 250 to 300 billion USD each year and 42 million life years were lost in disasters from 1980 to 2012. In spite of the technological and scientific advances in recent decades, it is still a challenge to accurately forecast the onset and/or advancement of many natural disasters. The main issues are as follows:

(i) Inaccuracy in observational data

(ii) Limitations of forecast models

Inaccuracy in observational data can be attributed to data errors, missing data and conflicting observed data. Core limitations in forecast models are related to the incomplete understanding and modelling of the underlying science, e.g. the physics, and the simplified numerical representations of complex processes. To tackle the impact of both issues, the following solutions are proposed:

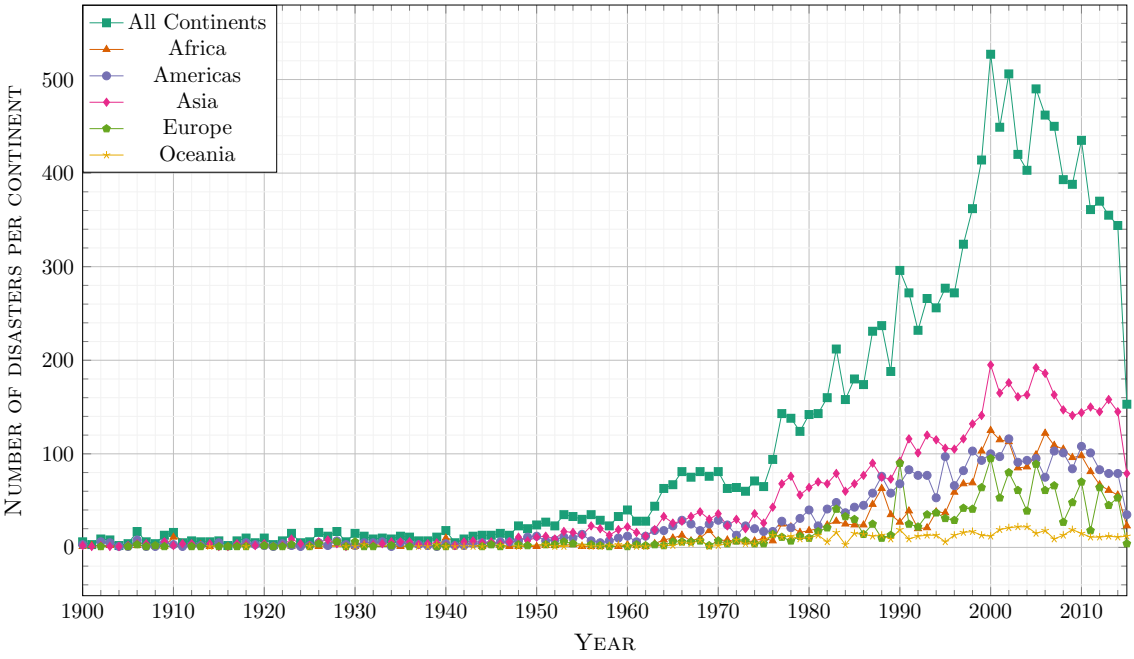
(i) Leverage on zero hour (shortly before/when/after the disaster strikes) data

(ii) Utilise stochastic forecast methods

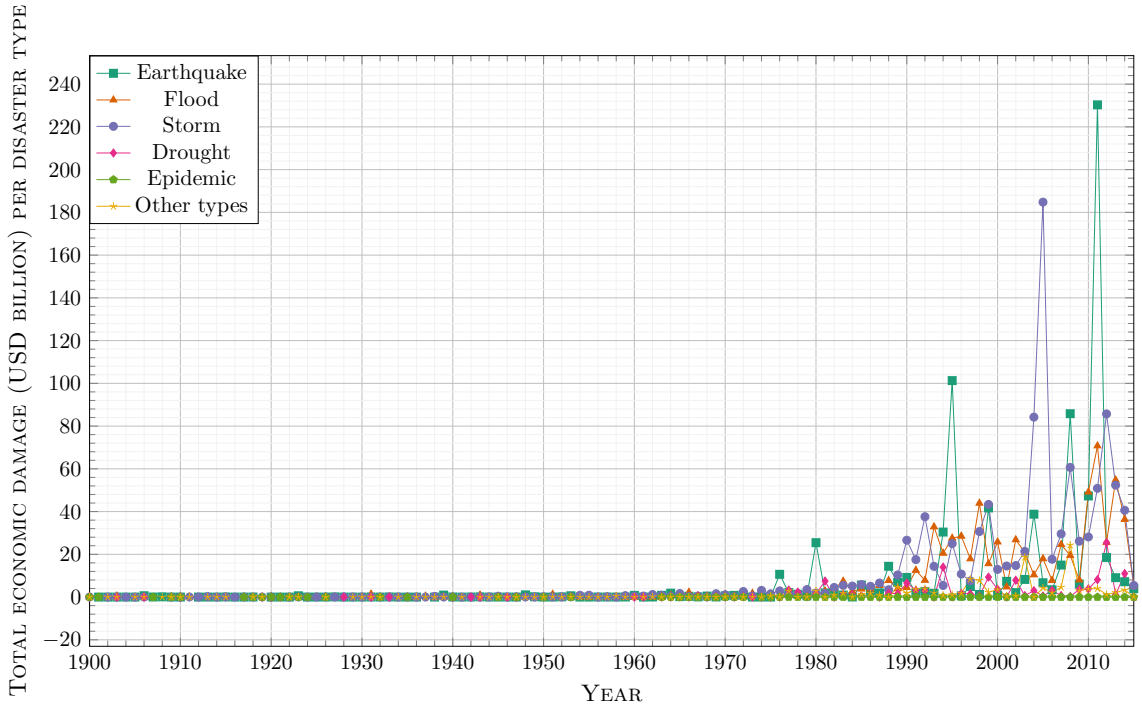
The accuracy of observational data is improved when more data is collected as the disaster approaches or develops. Figure 1.2 is redrawn from the figure[2] provided by National Oceanic and Atmospheric Administration (NOAA) and National Weather Service (NWS). It illustrates the relationship between the forecast confidence (x-axis) and forecast lead time (y-axis). Forecast *lead time* represents the amount of time left after computing the forecasts for issuing warnings and carrying out mitigation activities. The forecast lead time in the time scope from years to minutes (gray rectangular boxes) is illustrated. The longer the lead time, the lower the forecast confidence.

The forecast uncertainty is also illustrated in Figure 1.2 with the space between the two curved lines for a specific set of forecast confidence and forecast lead time. The forecast uncertainty reduces as the forecast lead time reduces and the forecast confidence increases.

[2] http://www.crh.noaa.gov/Image/lmk/pdf/WeatherForecastUncertainty.pdf



(a) Total Number of Reported Natural Disasters (EM-DAT)



(b) Total Economic Damage caused by Reported Natural Disasters (EM-DAT)

Figure 1.1: Disaster Trend between 1900 to 2015

When only a few days, hours and/or minutes of lead time are left, the forecast uncertainty can potentially be completely eliminated. The forecast confidence becomes correspondingly high. Forecast confidence is thus achieved at the expense of lead time by leveraging on zero hour data to improve the accuracy of forecasts.

In this dissertation, we are interested in the high forecast confidence forecasts with zero or low forecast uncertainty as illustrated in the circled area in Figure 1.2. *Zero hour data* represents the most up-to-date input data that is only available shortly before/when/after the event strikes.

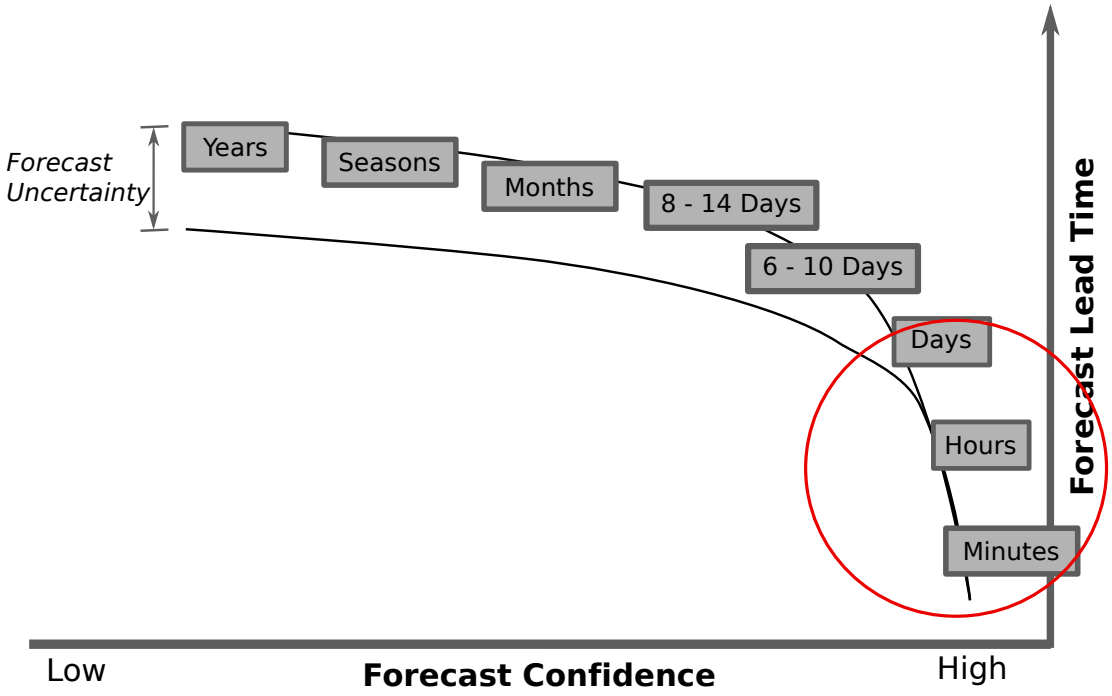


Figure 1.2: Forecast Confidence versus Lead Time (NOAA/NWS)

However, zero hour data alone is insufficient to ensure a high fidelity forecast. The limitations of forecast models have to be tackled too. Stochastic prediction methods [5] where an ensemble of forecasts [6] is leveraged upon for probabilistic forecasting is thus adopted as opposed to a deterministic approach.

Figure 1.3 illustrates an ensemble forecast of storms that began on 24 December 1999. It shows the surface pressure results of a forecast using a deterministic prediction method (first map from top left), the actual verification (second map from top left) and fifty different forecasts (bottom five rows of the maps) at 42 hours after the event commenced. The "Deterministic Prediction" indicated that no storms were expected in France and Germany but in reality as illustrated by "Verification", both countries were significantly hit by the storms. The ensemble of forecasts each depicts a different forecast with a varying intensity and spatial scope. Forecast 50 for instance indicates that the storm would likely only affect a very small area shown in the map. Forecast 41 however shows otherwise and

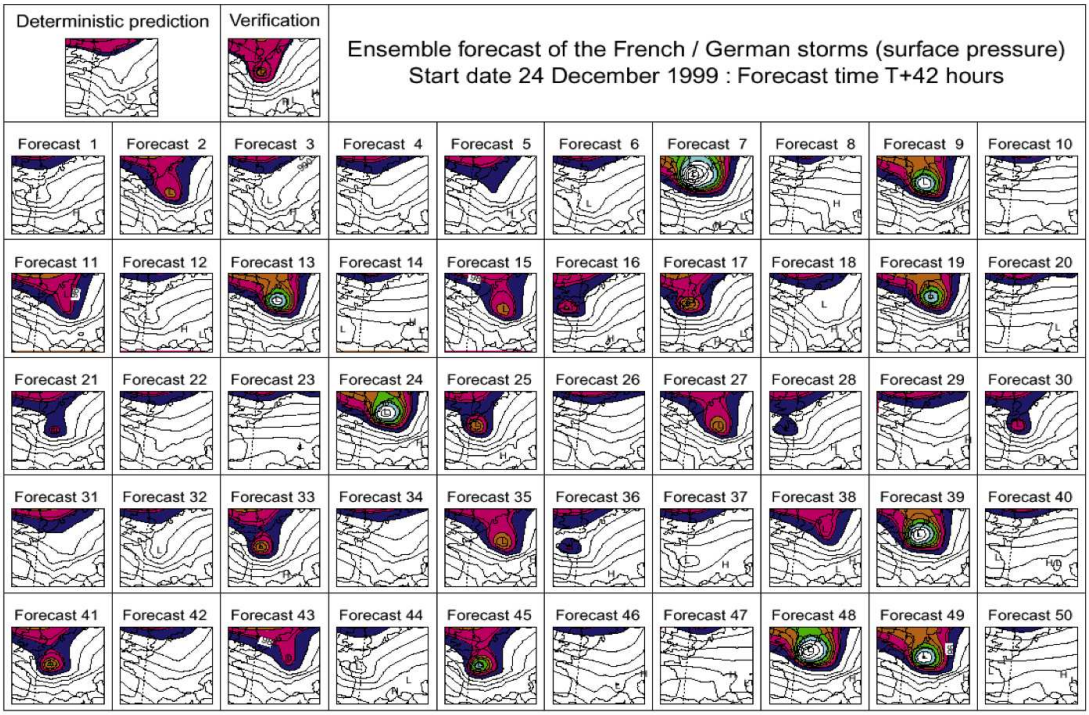


Figure 1.3: Ensemble Forecast of a Storm with 50 ECMWF Model Runs [1, p. 431]

is arguably one of the most accurate forecasts among the ensemble. What was particularly insightful in this case was that almost 50% of the ensemble of forecasts predicted that the storms would arrive in France and Germany, thus justifying the issuing of a storm warning. By utilising an ensemble of forecasts, the inaccuracy is distributed across all models and a higher fidelity forecast can thus be achieved.

To adopt both solutions, leveraging on zero hour data and utilising stochastic forecast methods, there is a need to confront the challenge in enabling multiple computational jobs simultaneously. These jobs have to run on one or more distributed heterogeneous computing resources while ensuring that the (short) stipulated deadline is met. Urgent computing, which targets time critical computations utilising data that are only available on short order to facilitate responsible authorities, e.g. civil defence services, to better understand a disaster and support in the making of educated decisions to manage affected areas to reduce casualties and losses, provides the research scope to solve this challenge.

Consequently, this dissertation aims to answer the following research question:

How to effectively enable urgent computing on a set of distributed heterogeneous resources while addressing the limitations of event domain sciences, uncertainty in forecast models and inaccuracy in observational data, by leveraging on an ensemble of forecasts that has to be computed before a stipulated deadline?

In order to answer the above question, the requirements and characteristics of urgent computing must be thoroughly understood. This dissertation thus also seeks a comprehens-

ive definition of urgent computing. It is the first known urgent computing work addressing ensembles of forecasts. Public infrastructures, which can potentially offer a wide array of computing resources, will be the main targeted class of resource.

1.2 Research Methodology and Contribution

The combined quantitative and qualitative research methodology adopted in this dissertation, a result of the natural process while pursuing the research activities, is summarised in Figure 1.4. The circular ring represents the four main research steps in the following order:

(i) Survey

(ii) Definition

(iii) Gather use cases

(iv) Ensemble of forecasts

The research goal, the Urgent Computing Framework, is depicted with a gray circle in the middle. The texts outer wrapping the respective research steps illustrate the corresponding research approaches and their findings. The bold texts inner wrapping the respective steps/approaches present the final research outcomes, which are then included in the Urgent Computing Framework. The detailed research activities based on this research methodology will be described in this section.

A (i) survey on urgent computing use cases [7] was conducted to gather the generic challenges and solutions. Based on the knowledge gained from the survey, the initial definition of urgent computing, offered by N. Trebon [8], was meticulously analysed. It was found to be too usage context specific and was thus unsuitable for general applications. To address the shortcomings, the (ii) definition is refined and extended [9]. Functional and non-functional requirements, an urgent system and its pre- and post-computation characteristics, deadline, cost and suitable classes of computing resources are additionally construed. Two particularly unique characteristics, non-deterministic deadlines and loss mitigation as opposed to loss prevention, are identified.

However, since urgent computing is dependent on the event (disaster) it is computing, the requirements from the event domain science are also required. Two groups of event domain scientists, hydro-meteorologists and seismologists, who were collaborating with Ludwig-Maximilians-Universität (LMU Munich) and Leibniz Supercomputing Centre (LRZ) in the EU projects, Distributed Research Infrastructure for Hydro-meteorology (DRIHM)[3] and Virtual Earthquake and seismology Research Community e-science environment in Europe (VERCE)[4], were consulted on the potential (iii) use cases. The

[3] http://www.drihm.eu/
[4] http://www.verce.eu/

qualitative approach was chosen due to the inherent complexity of the domain sciences. It requires an understanding of the underlying sciences, e.g. physics, to gain insights into their issues/problems and to obtain the opinions on the potential use cases. A questionnaire and an interview were thus conducted with each of the scientists. The questionnaire serves to elicit use case specific requirements, e.g. algorithmic solutions, suitability of different computing resources and computational requirements, while the interview captures the scientific requirements of the event domain science. This led to the very valuable insight on the common challenges, i.e. data inaccuracies and forecast models limitations, faced.

(iv) Ensembles of forecasts have thus to be incorporated to manage the uncertainty and improve the forecast confidence. The underlying framework has to coordinate multiple, typically tens of, forecasts simultaneously in order to meet the (short) deadline. Additionally, each forecast is a computing job which can utilise hundreds to thousands of cores on a parallel computing resource. This increases the computation requirements and illustrates a need to integrate distributed heterogeneous resource sets. The reliability of a system is improved by having distributed resources since "a single point of failure" can be avoided. Heterogeneous resources are expected as it is highly unlikely to find multiple distributed resources that are homogeneous in both hardware and computing environments.

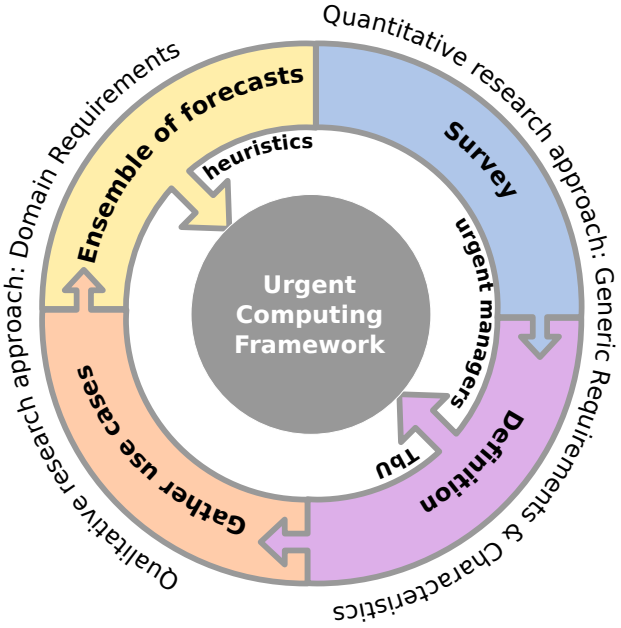


Figure 1.4: Research Methodology

In order to address the identified requirements and challenges, an urgent computing framework with a three-layer architecture was designed. This architecture allows a separation of concerns between the requirements and challenges, and the use case specific processes such as the pre- and post-processing. Four urgent software component managers, information, resource & environment, schedule and fault, are incorporated to administer the requirements, challenges and processes. To realise each of the managers, an analytical

approach consisting of a process model as shown in Figure 1.5, is employed. This model is composed of three phases in the following order, data collection, criterion function and utility function, with reference to the analytic hierarchy process (AHP) [10] where the decision-making processes of the manager are organised based on decision objectives, criteria and constraints in a hierarchy. Each phase aims to simplify the work processes by reducing the amount of information that has to be processed so that they can be carried out swiftly. The identification, collection and assessment of data based on the decision objective is the initial phase to enable each manager to gather relevant information on the prevalent state of a given resource set and/or user requirements. The second phase, criterion function, refines the data based on the mandatory criteria/requirements that the manager is entrusted to satisfy. In the final phase, the utility function performs a filtering of the data to find all the possible alternatives which might satisfy the defined non-mandatory constraints. The use of this model will be elaborated in Section 4.2.1.

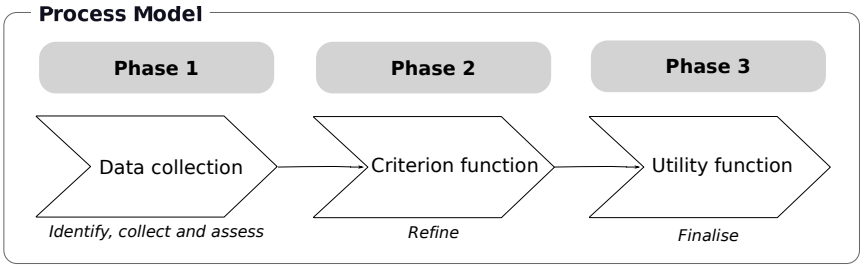


Figure 1.5: Process Model

The need to access the framework from anywhere and at anytime, leads to the construction and implementation of a Task-based Ubiquitous (TbU) approach [11] on top of the architecture. This approach facilitates access to the underlying distributed resource sets from ubiquitous end user devices, which have the highest probability of being readily available in event of a disaster. Backup resources that are spatially distributed are also crucial in the chaotic environment which entails a disaster since the inherent unpredictability of disasters can render any best-made plans futile, e.g. with a power failure. Finally, the ability to swiftly organise resources for an ensemble of forecasts such that the stipulated deadline can be met is of utmost importance.

The coordination of multiple heterogeneous distributed resources for an ensemble of forecasts before a deadline is realised by introducing a set of resource allocation heuristics. The heuristics have to ensure that the following two obligations are realised.

(i) Meeting the stipulated deadline

(ii) Maximising the number of successfully allocated and completed forecasts

Additionally, three objectives that have to be optimised by the heuristics to satisfy the requirements of urgent computing and the constraints of the targeted class of resources (refer to Chapter 5 and 6) are identified. The objectives are as follows:

(i) Makespan robustness

(ii) Reliability

(iii) Energy-To-Solution (ETS)

Makespan robustness refers to the tolerance of an urgent system to perform faithfully in event of perturbations such that the length of the schedule of a resource allocation adheres to the deadline. Maximising makespan robustness will implicitly imply minimising the makespan of an allocation, which will result in more lead time for mitigation activities. This is the most important objective among the three, since late results are useless as mitigation activities can no longer be carried out. This objective is thus fulfilling the most crucial timing criterion of urgent computing while optimising a resource allocation such that the lead time is maximised. In order to effectively quantify makespan robustness in urgent computing, a mathematical model to define the robustness radius and metric is developed.

Reliability aims to address the challenge of utilising multiple heterogeneous distributed computing resources for the ensembles of forecasts. Reliability of a resource allocation can be improved by avoiding a single point of failure, i.e. by not allocating all forecasts to only one resource or one resource site, and by selecting resources that will enable a maximum number of forecasts to successfully complete before the deadline. The aim of the reliability objective in a resource allocation is thus to select reliable resources for the forecasts and to distribute them among many such resources across multiple sites. A mathematical model to quantify the reliability, i.e. reliability radius and metric, across resources and computing sites is formulated.

Finally, the last objective, ETS, takes into consideration the realistic energy constraint of computing resource providers in supporting such computations. *ETS* is the total energy required for a computation to arrive at its solution. Frequency scaling options (the use of higher processor clock frequencies at the expense of energy), are now a commonality in modern high performance computing (HPC). By using higher frequencies, the makespan of an allocation can be significantly reduced. However, as the amount of energy required to power the computing resources is steadily increasing. This pushes the limits of resource providers, particularly the public ones, to foot the power bills and in some cases to operate within the power capacity limits of their centres. Thus the frequency scaling options are not available to users by default. Additionally, as the computing budget that each resource provider can dedicate to urgent computing is fixed and limited, the more energy efficient an allocation is, the more urgent computations each provider can support. There is thus a need to minimise the energy usage when appropriate.

An assessment model with reference to the identified obligations and objectives is designed to evaluate and compare the results of the allocation among the heuristics. The heuristics will function on the implicit requirement that the selected resources in a given resource allocation have sufficient free computing nodes/cores to start the assigned computations immmediately or in short order. However, this is typically not the case if advance reservations (long-term) or preemption of running jobs are not allowed. The project

SPRUCE (Special PRiority and Urgent Computing Environment) [12] has demonstrated the difficulties of resource providers in supporting preemption due to policy restrictions [7]. Thus the cost of preemption [13] on a top-tier production HPC resource is studied. Two cost models, least cost and least disruptive, were developed. The cost of preemption is then compared to the cost of some disasters that occurred in the last decade. Preemption cost is shown to be insignificant as compared to the economic losses caused by these disasters. Therefore, the benefit far outweighs the cost to support preemption on public HPC resources. To facilitate the adoption of preemption into the usage policy of such HPC centres, a set of policy recommendations is also provided.

In summary, to answer the research question in this dissertation, a combined qualitative and quantitative research methodology is used to uncover the requirements and challenges of urgent computing and event domain sciences. Consequently, a comprehensive definition of urgent computing, an urgent computing framework with a three-layer architecture, four urgent software component managers, a TbU approach, a set of robust, reliable and energy-aware heuristics to allocate ensembles of forecasts to multiple computing resources, two mathematical models to quantify the robustness and reliability in the heuristics and an assessment model to evaluate and compare the allocation results among the heuristics are presented. A study to evaluate the cost of preemption was carried out to help justify the use of public HPC infrastructures for urgent computing.

1.3 Publications

The publications published in the context of this dissertation are arranged in chronological order based on the submission dates. The first list consists of work where the content contributes directly to the research findings of the dissertation. The second list involves collaborations with domain scientists, seismologists, and workflow computer scientists, which influence the research direction undertaken within the dissertation.

1.3.1 Personal and Main Contributions

As the first author, this list of publications illustrates not only how the research is conducted but also the achievements on each step of the way.

(i) S. H. Leong, A. Frank and D. Kranzlmüller. "**Leveraging e-Infrastructures for Urgent Computing**" [7]— This conference paper attempts to provide a survey and carry out an evaluation of the technical challenges faced and the improvements required to support urgent computing on existing e-Infrastructures in order to identify what are lacking and what are already available. The co-author, A. Frank, contributed by sharing his view on available practical policy issues faced by LRZ, a resource provider, which is a member of two European e-Infrastructures, Partnership for Ad-

vanced Computing in Europe (PRACE)[5] and European Grid Infrastructure (EGI)[6]. The findings of this paper are presented in Section 2.1.3.

(ii) S. H. Leong, D. Kranzlmüller and A. Frank. "**A data management system to enable urgent natural disaster computing**" [14]— This poster presents an initial design and implementation of the urgent computing framework as a data management system. The co-author, A. Frank, contributed once again by sharing his view on practical policy restrictions that can influence the success of such a system. This initial setup helps in defining the architecture of the urgent system as shared in Chapter 4.

(iii) S. H. Leong and D. Kranzlmüller. "**Advance Visualisation and Urgent Computing**" [15]— This short paper evaluates the compatibility of advance visualisation and urgent computing. The current shortcomings in advance visualisation are highlighted since they conflict with the deadline criterion of urgent computing. However, the importance of visualisation to assist in evaluating the computed data is also emphasised.

(iv) S. H. Leong and D. Kranzlmüller. "**Towards a General Definition of Urgent Computing**" [9]— This conference paper aims to extend and refine the existing urgent computing definition to provide a comprehensive general version to aid in the identification of its unique challenges. The requirements of urgent computing, the urgent system and its characteristics, deadline, cost and classes of computing resource are elaborated in Chapter 3.

(v) S. H. Leong and D. Kranzlmüller. "**A Case Study - Cost of Preemption for Urgent Computing on SuperMUC**" [13]— This conference paper investigates the cost of preemption on a petaflop operational HPC resource, SuperMUC, by using two specially designed and implemented cost algorithms, least cost and least disruptive. It demonstrates that the cost of preemption is in fact much lower in comparison to the loss mitigation that can be achieved by allowing an urgent computation. The ultimate aim is to provide evidence to convince policy makers on the feasibility and benefits of supporting urgent computing on public resources. The cost equation defined in [9] formed the basis for computing the cost of preemption on SuperMUC. Consequently, the resource allocation heuristics proposed in [16] are designed on the assumption that preemption can be realised.

(vi) S. H. Leong and D. Kranzlmüller. "**A Task-based Ubiquitous Approach to Urgent Computing for Disaster Management**" [11]— This conference paper focuses on elaborating the urgent computing framework, a three-layer architecture, based on a task-based ubiquitous approach where four urgent software compon-

[5]http://http://www.prace-ri.eu/
[6]http://www.egi.eu/

ent managers are also introduced to administer the urgent computing requirements defined in [9] and are described in Chapter 4.2.

(vii) S. H. Leong and D. Kranzlmüller. "**A Hydro-meteorological Urgent Computing Ubiquitous Framework for an Ensemble Forecast**" [17]— This short paper shows the first attempt to elaborate the entire urgent computing framework by illustrating the TbU approach, the urgent managers and the robust, reliable and energy-aware resource allocation heuristics as one urgent computer system.

(viii) S. H. Leong and D. Kranzlmüller. "**Urgent Computing - A General Makespan Robustness Model for Ensembles of Forecasts**" [18] — This conference paper presents a general mathematical makespan robustness model for urgent computing to allocate ensembles of forecasts to computing resources such that small perturbations in the resources will not adversely influence the ability to meet the stipulated deadline. It illustrates the need for additional objectives, e.g. reliability, to enable an effective resource allocation. Consequently, a follow-up paper [16] is prepared.

(ix) S. H. Leong and D. Kranzlmüller. "**An Urgent Computing Framework for Ensembles of Forecasts on HPC Infrastructure**" [19] — This short paper presents the final design and implementation of the urgent computing framework based on the TbU approach for an ensemble of forecasts. The chosen heuristic from [16] will be included in one of the four urgent software component managers, the schedule manager, as defined in [11].

(x) S. H. Leong, A. Parodi and D. Kranzlmüller. "**A Robust Reliable Energy-Aware Urgent Computing Resource Allocation for Flash Flood Ensemble Forecasting on HPC Infrastructures for Decision Support**" [16] — This journal paper shares a set of robust, reliable and energy-aware allocation heuristics for ensembles of flash flood forecasts to tackle the requirements and challenges of urgent computing and the event domain sciences. The co-author, A. Parodi, contributed by sharing his expert knowledge on hydro-meteorology, the flash flood problem and provided an expert analysis of the visualised ensemble results. The makespan robustness model in [18] is extended to include frequency scaling, which can improve the makespan of an allocation. A reliability model to quantify site and resource reliability is also developed to support a good distribution of the forecasts on reliable resources across computing sites and resources. ETS is also included in the heuristics to ensure that the realistic power constraints of resource providers are also taken into consideration. An assessment model to evaluate and compare the allocation results among the heuristics is also introduced. The robustness and reliability models, the set of proposed heuristics and the assessment model will be shared in Chapter 5.

1.3.2 Collaborative Contributions

This list of publications contains mainly publications from the European Union's Seventh Framework Programme project, VERCE. All publications are a collaborative effort between the event domain scientists, i.e. seismologists, and computer scientists who specialised in either workflow management or infrastructure integration. As an infrastructure specialist, my main contributions in all associated papers revolve around enabling and integrating the defined seismological use cases and corresponding workflows effectively on European e-Infrastructures.

(i) M. Carpene, I. Klampanos, S. H. Leong, E. Casarotti, P. Danecek, G. Ferini, A. Gemnd, A. Krause, L. Krischer, F. Magnoni, M. Simon, A. Spinuso, L. Trani, M. Atkinson, G. Erbacci, A. Frank, H. Igel, A. Rietbrock, H. Schwichtenberg and J.-P. Vilotte. "**Towards Addressing CPU-Intensive Seismological Applications in Europe**" [20]— This conference paper illustrates the VERCE platform for the forward simulation use case that is to be integrated to the European e-Infrastructure, PRACE and EGI. It facilitates the identification of the common pitfalls that ultimately contributes to the design and implementation of the urgent system architecture and TbU approach.

(ii) M. Simon and S. H. Leong and K. H. Zad and L. Krischer and M. Carpene and G. Ferini and L. Trani and A. Spinuso and F. Magnoni and E. Casarotti and A. Gemünd and D. Weissenbach and I. Klampanos and H. Igel. "**VERCE - CPU-intensive Applications in Seismology**" [21]— This short paper shares the initial platform as a model framework that has the potential to fulfil different use cases on various e-Infrastructures like PRACE and EGI by leveraging on the open standard, A Simple API for Grid Applications (SAGA) [22].

(iii) A. Spinuso, A. Krause, C.R. Garcia, E. Casarotti, F. Magnoni, I.A. Klampanos, L. Frobert, L. Krischer, L. Trani, M. David, S. H. Leong, V. Muraleedharan. "**The VERCE Science Gateway: enabling user friendly seismic waves simulations across European HPC infrastructures**" [23]— This short paper describes the technological choices within the VERCE Platform to enable the Science Gateway to integrate seamlessly with various EU e-Infrastructures, including EUDAT[7].

(iv) A. Spinuso, A. Krause, C.R. Garcia, E. Casarotti, F. Magnoni, J. Matser, L. Krischer, L. Trani, M. David, S. H. Leong and V. Muraleedharan. "**The VERCE Science Gateway: Interactive Forward Modeling and Metadata Management**" [24]— This short paper shares the common processing elements (PEs), data and web services, utilised within the forward modelling workflow. Provenance data, i.e. meta-data, is created, stored and can be browsed, queried and retrieved from the gateway.

[7] http://eudat.eu/

(v) E. Casarotti, A. Spinuso, J. Matser, S. H. Leong, F. Magnoni, A. Krause, C. R. Garcia, V. Muraleedharan, L. Krischer and C. Anthes. "**The VERCE Science Gateway: Enabling User Friendly HPC Seismic Wave Simulations**" [25]— This short paper elaborates on how the VERCE Science Gateway enables the easy setup of the forward modelling problem and to compute it on EU e-Infrastructures from a seismologist's point of view. It also shares the visualisation achievements within the project.

(vi) S. H. Leong, C. Anthes, F. Magnoni, A. Spinuso and E. Casarotti. "**Advance Visualisation of Seismic Wave Propagation and Speed Model**" [26]— This article illustrates the advance visualisation of a seismic wave propagation and speed model in the Virtual Reality installations, i.e. a 5 sided projection installation that is based on the concepts of a Carolina Cruz-Neira's CAVE Automated Virtual Environment (CAVE)[8].

(vii) M. Atkinson, M. Carpene, E. Casarotti, S. Claus, R. Filgueira, A. Frank, M. Galea, A. Gemünd, H. Igel, I. Klampanos, A. Krause, L. Krischer, S. H. Leong, F. Magnoni, J. Matser, A. Michelini, H. Schwichtenberg, A. Spinuso and J.-P. Vilotte. "**VERCE delivers a productive e-Science environment for seismology research**" [27]— This conference paper presents the motivation for building the VERCE platform to support solid-Earth scientists to use established simulation codes on HPCs in conjunction with multiple sources of observational data, which can contribute significantly to their researches. The architecture of the VERCE platform and the underlying mechanisms are also shared.

1.4 Outline

The dissertation is organised as follows:

- Chapter 2 - Related Work & Evaluation
  An overview of the related work in urgent computing and the proposed solution in this dissertation is discussed in this chapter. A summary of urgent computing research activities from 2006 until now is shared. The original definition of urgent computing is also scrutinised to identify its shortcomings. Computing paradigms, e.g. real-time computing, which share similar characteristics with urgent computing are outlined. The importance of adopting a stochastic approach, i.e. an ensemble of forecasts in the framework, is explained. Research activities in resource allocation, in particular heuristics, are also elaborated.

- Chapter 3 - Urgent Computing Definition
  In this chapter, the shortcomings in the original definition of urgent computing identified in Chapter 2 are extended and refined to provide a comprehensive general

[8] CAVE™, a registered trademark of the University of Illinois' Board of Trustees

version. The updated version includes a clarification on the requirements and characteristics, i.e pre-computation and post-computation characteristics, deadline and cost. The target classes of resources for urgent computations are described here. A set of policy recommendations to enable urgent computing on shared resources is also presented.

- Chapter 4 - An Urgent System & A Task-based Ubiquitous Framework
  An urgent system consisting of four main components, the operator, the urgent computer system, the decision & coordination system and controlled objects, is defined in this chapter. A task-based ubiquitous framework based on a three-layer architecture is presented to enable easy accessibility to the system in chaotic times when a disaster is expected or occurring. The framework also offers the flexibility to adapt the system according to dynamic use case specific requirements on heterogeneous resources. Four urgent software component managers, information, resource & environment, schedule and fault, are designed to manage the requirements of urgent computing.

- Chapter 5 - Resource Allocation Heuristics
  One of the most crucial requirement, deadline, is realised in this chapter by introducing a set of resource allocation heuristics that will be a part of one of the urgent software component managers, the schedule manager. Two obligations and three objectives, robustness, reliability and effective energy usage, of a resource allocation for ensembles of forecasts are identified. Three resource allocation patterns are shared to establish the most effective pattern for allocating resources to ensembles of forecasts. Two mathematical models are specially implemented to quantify reliability and robustness. Energy-to-solution (ETS) is used to assess the energy usage of a computation on a specific resource. An assessment model is also designed to organise and evaluate the results of an allocation and to compare the results among different allocations.

- Chapter 6 - Implementation and Result
  In order to study the effectiveness of the designed heuristics, two case studies are carried out. The first case study attempts to evaluate the three forecast allocation patterns to identify the most time effective forecast pattern for an ensemble of forecasts. The second case study investigates a real flash flood that occurred in October 2014 in Genoa. A set of resource allocation heuristics is designed and leveraged upon to schedule an ensemble of forecasts on a set of heterogeneous distributed resources using the identified forecast pattern. Multiple objectives, i.e. reliability, robustness and efficient ETS, are corresponding fulfilled in a balanced manner. The required compromise among different heuristics is shared and the most optimal heuristic is identified.

- Chapter 7 - Conclusion and Future Work
  The work of this dissertation is summarised and concluded in this chapter. Potential

future works that are closely related to this dissertation are proposed.

1.5 Terminology

Common terms used while defining urgent computing are clarified in the following list.

- An **urgent use case** is a description of a recurring issue, e.g. a flood, or a high impact issue, e.g. a nuclear meltdown in a reactor, which is expected to potentially result in extensive loss. The economic loss and human casualties are expected to be mitigated with the support of time-critical computations.

- An **urgent event** is an occurrence at a point in space and time that can potentially create an extensive loss situation, which requires immediate attention. An urgent use case is thus a description of an urgent event.

- An **urgent service** is the act of the activities, i.e. computation, decision making and coordination work, to fulfil the functions of an urgent system.

- An **urgent computation** is a computing activity that must commence in short order, i.e. immediately or as soon as possible, to forecast the commencement and/or progress of an urgent event. An urgent computation is triggered by an urgent event, either the expectation or the commencement of it.

- An **urgent product** is the processed result of an urgent computation that can be directly used for decision making and coordination activities to support loss mitigation.

- An **urgent computer system** is a component within an urgent system that is in charge of the urgent computations. The computing resources are a part of this component.

- An **urgent system** is a system to enable urgent computation(s) and support decision making and coordination work such that loss mitigation is possible. The term "urgent system" is selected as opposed to "urgent computing system" to take the cue from real-time computing, where the system is known as "real-time system" as opposed to "real-time computing system".

- A **lead time** refers to the time left after the urgent computations are completed for mitigation activities.

- **Makespan** is the length of a schedule of a resource allocation for the urgent computation(s). It represents the time from the start of the allocation to the point in time when all assigned computations complete.

- An **event domain scientist** in this dissertation refers to a domain specialist who is an expert in a specific urgent event and can thus decide if an urgent computation should be initiated and has the knowledge to evaluate urgent products to make recommendations to decision makers. For convenience, an event domain scientist is at times simply referred to as a domain scientist in this dissertation.

Finally when referring to the rows and columns, e.g. the first row, of a table, the header row and column are excluded from the count.

Chapter 2

Related Work and Analysis

The concepts behind urgent computing rely on a mixture of disciplines, services and policies. As such, a variety of related work has to be evaluated. This chapter will begin by providing an overview of urgent computing in Section 2.1 where the background of the first known urgent computing project is shared. Next, related computing paradigms, Real-Time Computing, Crisis and Disaster Management Computing and On-Demand Computing, which have similarities to urgent computing are presented. This is followed by a description and analysis of the existing and only definition of urgent computing from Trebon [8]. Finally, the compiled results of existing urgent computing use cases and challenges, and two urgent computer systems related to TbU are discussed. This section will help in extending the definition of urgent computing as shown in Chapter 3 and assist in the design of the architecture of an urgent system and the TbU approach in Chapter 4.

Section 2.2 describes the related work in resource allocation and resource allocation heuristics. A set of resource allocation heuristics specifically designed for urgent computing is elaborated in Chapter 5. Section 2.3 provides an overview of the worldwide adoption of ensemble forecasting in weather prediction. The importance of the selected use case, flooding, which is used to evaluate the solution presented in this dissertation (refer to Chapter 6) is also shared.

2.1 Overview of Urgent Computing

Urgent computing first came into limelight in 2006 via the US project, SPRUCE (Special PRiority and Urgent Computing Environment) [12]. Before that, urgent computations are simply thought as time-critical computations, which can be categorised under e.g. real-time computing, without carefully considering its unique requirements and characteristics. Since many of the use cases require significant computation power, e.g. HPC, in order to

maximise the lead time, it was avoided and less time-critical forecasts utilising data from days to weeks in advance were more common. SPRUCE provides an urgent computer system to access an array of HPC resources within TeraGrid, predecessor project of Extreme Science and Engineering Discovery Environment (XSEDE)[1]. It is arguably the most ambitious and complete urgent computing initiative ever attempted with the aim to support urgent use cases from multiple event domain sciences.

An elevated priority, preemption and next-to-run, solution was implemented in SPRUCE to handle urgent jobs by using the *Right-of-Way* tokens. Finer details were also considered by having procedures to prepare in advance the required scientific applications on the resources, easy to access web portals for administrators and users, etc. However, several limitations hindered the success of SPRUCE. Preemption is hard to realise in the production mode since existing policies make it impossible for resource providers to pause or delete running jobs from the system to make way for urgent computations, as this would violate the providers' existing usage policies. Consequently, there was only one provider in SPRUCE that supported preemption. The next-to-run strategy was more widely adopted but could delay the urgent computations while waiting for running jobs to finish. There is a known use case [28] that did not successfully leverage on the SPRUCE implementation. The North Carolina storm surge forecast started by using the SPRUCE software stack before switching to a dedicated computing system at RENCI[2]. Another development was that when TeraGrid was ending, there was no longer any resource that supported preemption [28].

SPRUCE faced another major challenge towards the end of its funding period due to the pitfall in dynamism of computing environments. In order to ease the access to the different resources in TeraGrid, an external open-source middleware tool, Globus Toolkit (GT)[3], was selected as the common toolkit to provide the required services. As a middleware, GT hid the dynamism of computing environments by offering generic access interfaces for many different services, e.g. interactive access, resource monitoring and discovery, and data transfer management, across resources. This simplified the complexity in providing urgent computing software tools and services within SPRUCE. The over-reliance on one middleware however led to an unfortunate pitfall. During an upgrade in 2009, the developers of GT decided to make a major technology switch, rendering many of the crucial services required by SPRUCE obsolete. Consequently, SPRUCE attempted to leverage on on-demand resources, by beginning work to incorporate computational clouds as urgent computing resources in the same year[4]. However due to the small set of offered cloud resources and the fact that parallel computing was not supported on the cloud offered within SPRUCE [28], many urgent use cases could not leverage on the provided cloud infrastructure.

Cloud has since became the prevalent urgent computing resource choice, e.g. an urban flood decision support system [29] and an early warning workflow-based framework [30].

[1] https://www.xsede.org/
[2] http://renci.org/
[3] http://toolkit.globus.org/
[4] http://spruce.teragrid.org/news.php

It is a highly valuable class of resources for use cases that do not require the computing capability of expensive public HPC resources. The cloud is readily available, flexible to configure and easier to maintain as a homogeneous environment that is independent of cloud providers. This is a direct contrast to public HPC resources, where access policy, security, specific environmental setup, etc. are challenges one has to contend with. However, there is a significant number of public HPC resources that have more computation capabilities than any cloud providers, and can thus realise a wide array of urgent use cases. They can also be significantly cheaper than commercial on-demand HPC resources as demonstrated in Section 3.5.1.

2.1.1 Related Computing Paradigms

Urgent computing is frequently being confused with other computing paradigms, in particular real-time computing (RTC), and crisis and disaster management computing (CDMC). The timing characteristics of urgent computing brings to mind RTC while its application, to manage loss from urgent events, hints a relationship to CDMC. Consequently, it prompts some to propose inappropriate (near) real-time solutions for urgent computing challenges. Non-time critical CDMC computations were also mistaken as urgent computations. On-demand computing (ODC) is another paradigm, which is commonly associated with urgent computing due to the requirement to have computing resource(s) on demand in short order. This confusion among the paradigms obstructs the identification of unique challenges and use cases, and impedes the progress of urgent computing. Thus, the differences among urgent computing, RTC, CDMC and ODC will be presented in this section.

Real-time Computing

The use of computers for "real-time" application was first proposed in 1950 by Brown and Campbell [31]. In 1954, the first digital computers were developed for a real-time system to provide an automatic flight and weapons control system. By 1974, a distributed computer control system was made possible. In comparison to urgent computing, the research effort and achievement in real-time systems is extensive due to its longer history. Understanding the relationship between urgent and real-time systems will be greatly beneficial in shaping the definition, characterisation and modelling of urgent systems.

Real-time computing is a computer science area where hardware and software systems, e.g. brake systems in a automotive and air traffic control systems, are subjected to real-time constraints. Although there is no single official definition of real-time system, the following two definitions illustrate the most crucial characteristics of real-time computing.

> "A system that performs its functions and responds to external, asynchronous events within a predictable (or deterministic) amount of time." [32]

> "The correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced" [2, p. 10]

The time constraint in real-time computing is annotated as a *deadline*, after which the results are not longer useful even if they are correctly produced. Three type of deadlines are defined in real-time systems, soft, firm and hard. A real-time system with a soft, firm or hard deadline is correspondingly known as a soft, firm or hard real-time system respectively. The definitions of the real-time deadlines as given by [33, p.7] are illustrated with the following classifications.:

- A deadline is *soft* if the utility of results produced by a task decreases over time after the deadline expires.
- A deadline is said to be *firm* if the results produced by the corresponding task cease to be useful as soon as the deadline expires, but consequences of not meeting the deadline are not very severe.
- A deadline is said to be *hard* if the consequences of not meeting it can be catastrophic.

Predictability is a crucial characteristic of a real-time system. Knowing the deadline is thus a pre-requisite. A deadline is either fixed in advance or dynamic (dependent on the event), but is known, i.e. predictable, when a computation is initiated. However, a deadline of urgent computing can be an unknown, i.e. unpredictable, and even a target result of an urgent computation. A real-time system is typically modelled for a specific use case and the system is carefully "designed" to maintain predictability under pre-defined conditions. In the case of urgent system, whether predictability can be "designed" into the system is dependent on the urgent event.

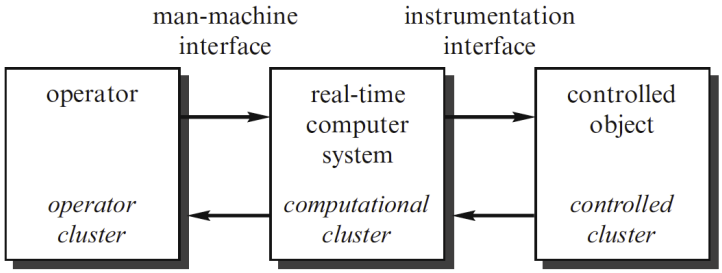


Figure 2.1: Real-time System [2]

A real-time system is represented as shown in Figure 2.1 by [2] where it consists of three building blocks, the operator, the real-time computer system and the controlled object. Interactions between the neighbouring blocks occurred over the respective interfaces. On the operator cluster, an operator can initiate a computation via the man-machine interface. The real-time computations are then initiated on the computational cluster. Finally, on the controlled cluster, the controlled object is precisely controlled, by consulting the computation result, via the instrumentation interface.

Figure 2.2 summarises the relationship between real-time, distributed real-time and urgent computing with a Venn diagram. The differences in the sizes of the circles aim to

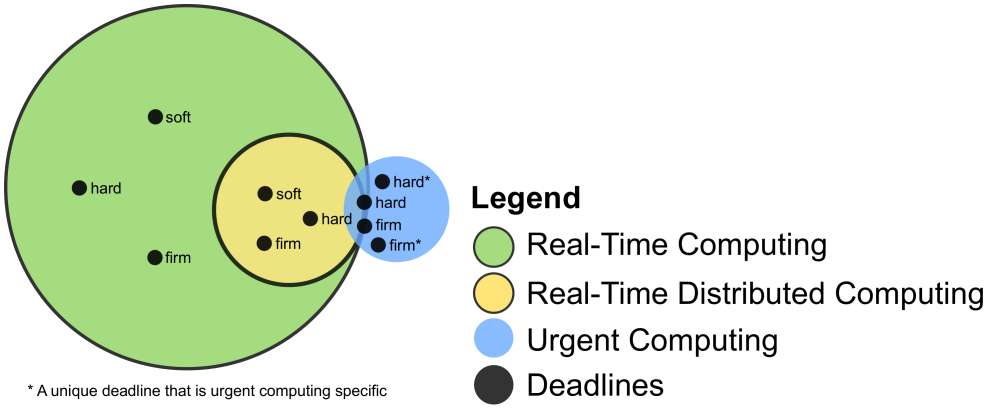


Figure 2.2: Venn diagram of Real-time Computing and Urgent computing

represent the differences in the amount of research activities within each classification of computing. A distributed real-time system, which consists of a set of (computer) nodes that are interconnected by a real-time communication network [2, p. 2], is a subset of a real-time system. As such, it has the same three classifications of deadlines, soft, firm and hard (depicted by small black circles). An urgent system can take advantage of both single resource and distributed resources (represented by the intersection among the three biggest circles). In spite of sharing many traits, an urgent system has some fundamentally distinct requirements and characteristics (refer to Chapter 3) and in particular its deadline classifications. An urgent system has only firm and hard deadline classifications. Additionally, it has two special firm* and hard* deadlines (refer to 3.3) that are uniquely urgent computing.

Crisis and Disaster Management Computing

Crisis management computing, disaster management computing, emergency computing, computing for disasters and a few more variants containing the key terms, civil protection, disaster, crisis and emergency, can be taken as synonymous in this context. Frequently the two terms, disaster management and crisis management, come hand-in-hand since a disaster can result in a crisis if it is sufficiently severe. In principle, CDMC includes the following activities:

(a) **Prevention** - Reduce the risk of a crisis

(b) **Preparation** - Prepare for a crisis

(c) **Response** - Respond to a crisis

(d) **Recovery** - Recover from a crisis

The term *crisis* will also be used to represent the threat that triggers one or more computations since the disasters, both natural and man-made, which urgent computing is

concerned with are those that will potentially result in a crisis. CDMC is thus a class of computation activities to manage the potential crises.

Computations to reduce the risk of a crisis can include any computing that facilitates a better understanding of a particular crisis in order to plan measures to reduce the possibility of its occurrence. This class of computation takes place before a crisis occurs. An example is the collection and computation of rainfall statistics in a high risk region to plan the depth of drainage in order to reduce the likelihood of a flood.

In many crises, there are no ways to prevent them, e.g. storm and tsunami, from happening but the damages can be readily reduced by being more prepared. These preventive activities can continue until the crisis strikes. Using the flood as an example, in regions where one cannot prevent a flood, computations to predict the rain depth can help in early evacuation and loss mitigation.

During a crisis, additional computations can be imperative for rescue work and loss mitigation. After a severe earthquake, there are typically strong aftershocks that can hinder rescue work and result in further losses. Computations to check the structural soundness of bridges can support the rescue work and protect rescue workers.

Finally, after a crisis, recovery activities must be carried out. An example of a recovery activity was the simulation of the anticipated path of the oil spill after the Deepwater Horizon oil rig explosion [34] in 2010.

The crisis management activities mentioned in the list above can thus be further categorised based on when they take place as shown in Figure 2.3. (a)Prevention and (b) preparation, are a part of pre-disaster activities. (c) Response and (d) recovery are a part of the in-disaster and post-disaster activities respectively.

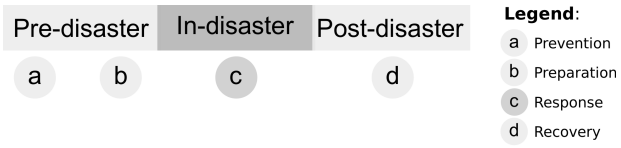


Figure 2.3: Crisis Management Computing Categorisation

The main difference between CDMC and urgent computing is the urgency factor. CDMC includes activities, particularly activities to reduce the risk of a crisis, which can have the luxury of time to be computed when the conditions permit. Urgent computing however includes only activities that have a bounded time constraint where waiting for computing resources to become available is infeasible.

Figure 2.4 summarises the relationship among the CDMC activities and urgent computing. Urgent computing can be seen as a subset of CDMC, which is concerned only with activities that have challenging time constraints. Its activities thus centre around some of the preparation activities (refer to gray circle (b)) that are happening shortly before the crisis (pre-disaster), all response activities (refer to gray circle (c)) during the crisis (in-disaster) and some of the recovery activities (refer to gray circle (d)) happening shortly after the crisis (post-disaster).

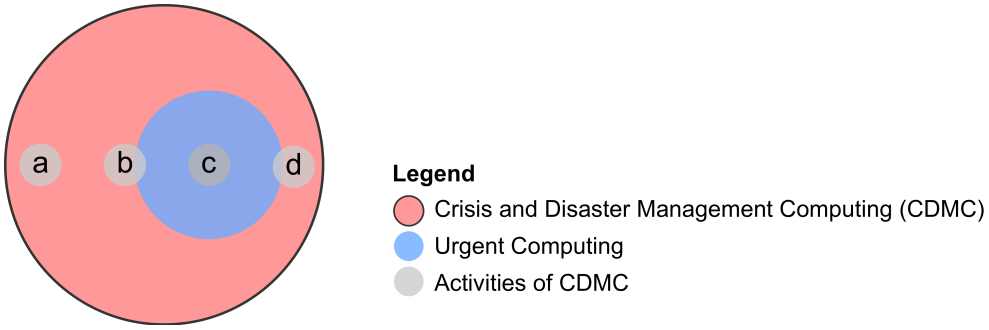


Figure 2.4: Venn Diagram of CDM Computing and Urgent Computing

On-Demand Computing

ODC is a class of computing where one does not purchase the computing resources but instead "leases" it on demand, usually with a cost, based on the computational needs. Cloud computing and utility computing are considered as implementations of this class of computing.

In order to realise urgent computing, resources have to be available on short notice, which is in line with one of the core features of on-demand computing. In contrast, shared public resources will require intrusive measures, e.g. preemption, which are likely to conflict with the operational policies of resource providers. On-demand computing is potentially cheaper than owning dedicated resources, particularly if it is leveraged upon by rarely occurring urgent events. It can serve as a backup resource for frequently occurring urgent events with dedicated resources, in the event of unforeseeable outage or planned maintenance.

Due to the ease of usage, there is an increasing dominance of cloud-based [29][30] systems for urgent computing. Such systems leverage on the fact that virtualisation enables the required computing environments to appear as homogeneous resources. Consequently the integration effort to computing resources is greatly simplified. On-demand computing should be regarded as a class of resource that is suitable and useful for urgent computing and in particular rarely occurring urgent events. Section 3.5 shares more about each class of computing resources.

2.1.2 Definition of Urgent computing

The first SPRUCE publication was a whitepaper [35] in 2006, describing SPRUCE's user workflow, portal, resource providers and implementation status. The first conference publication [12] from the same authors occurred in 2007 by focusing on SPRUCE as an urgent high performance computing system. The term "urgent computing" was introduced on the pretext that it was self-explanatory and was thus without a definition or reference. Four implementation requirements for supporting urgent computing were stated as follows:

(i) Job sessions

(ii) Policy framework among resource providers

(iii) Permission for priority jobs

(iv) User authorisation for priority jobs

The next paper [36] from 2008 clarified the needs and requirements of an urgent system, i.e. the SPRUCE system. The implementation requirements were further elaborated in the descriptions of the urgent computing framework. Urgent computing (system) was not formally defined but was hinted at with examples illustrating its usage, and keywords and phrases like "immediate attention", "event-driven", "deadline-based" and "late results are useless".

The first comprehensive and only definition of urgent computing was penned by Trebon [8] in his dissertation work to enable urgent computing within the existing distributed computing infrastructure, i.e. SPRUCE. Understandably, his work was influenced by SPRUCE's requirements, leading to a usage context specific definition of urgent computing. It imposed a number of restrictions in the requirements, which were considered as unnecessary and are discussed in detail in the following section.

Now, there are numerous urgent computing related publications for specific use cases [28][37], addressing specific challenges such as knowledge-based structures for simulation within computing tasks [38] and data management [14], and sharing an early warning system framework [30]. In these works, there were no attempts to refine the definition of urgent computing and use cases [39] that did not require significant resource usage or long computation time, i.e. violating the requirements defined by Trebon, were accepted as urgent use cases.

Trebon's Definition of Urgent Computing

There is thus a need to evaluate the existing definition of urgent computing and refine it such that it is applicable in a more general context while conforming with requirements of existing urgent work. Since the aim of Trebon's work was to design and implement an urgent computing environment that took advantage of the HPC resources within the TeraGrid, urgent computing was defined with the following three requirements [8, p.2]:

(req. 1) The computation operates under a strict deadline after which the computation results may give little practical value.

(req. 2) The onset of the event that necessitates the computation is unpredictable.

(req. 3) The computation requires significant resource usage.

This set of requirements provides a good hint to what urgent computing is but is too specific to SPRUCE's requirements and is thus unsuitable for general application. The shortcomings with each requirement is summarised respectively in the following list.

(sc. 1) The deadline classification, strict, is imprecise and too SPRUCE specific.

(sc. 2) The identification of onset as the characteristic of the events, which triggers urgent computations is inaccurate.

(sc. 3) The use of the term, significant, to quantify resource usage is imprecise.

In requirement **(req. 1)**, the deadline was defined to be "strict" but the term "strict" was not explained or defined. Instead Trebon shared that an urgent system falls somewhere between a soft and a hard real-time system [8, p.9], raising the question of whether a strict deadline corresponds to a firm deadline since it is the intermediate deadline classification between soft and hard.

Trebon was thus consulted via a written correspondence where he confirmed our suspicion. Since SPRUCE only requires the computations to aid in the decision making [8, p.11] as one of the information sources, the consequence of missing the deadline is expected to be not as severe, because mitigation activities can still be planned and carried out using other information sources. This led to the firm deadline classification. However, this is too restrictive for general application. In cases where decisions are only dependent on urgent computing as the only source of information, the deadline should be hard. Thus, both firm and hard deadlines should be included to have a complete urgent computing deadline classification.

The requirement **(req. 2)** is arguably the most insightful requirement among the three. It describes the unpredictable nature of an urgent event, albeit just the onset, and thus the rationale behind the urgency, i.e. time criteria, in urgent computing. However, rather than the unpredictability of the onset, the urgency is a result of the lack of or late arrival of more accurate data until zero hour. In case of a storm, the onset of the event is known but due to its dynamic and long lasting nature, urgent computations have to be performed as the event develops, i.e. as the most up-to-date data become available. This requirement must thus be extended to accurately reflect the characteristics of urgent computing and the cause behind the urgency of computations.

The requirement **(req. 3)** is set up such that the impressive array of expensive HPC resources, which SPRUCE had access to, can be efficiently utilised. In principle, there is no scientific or technical need to set a quantitative limit on the amount of resource usage. In fact, if a computation can use less cores/nodes while meeting the deadline, there is no reason not to accept it as an urgent computation. Trebon shared that it is simply more interesting, i.e. challenging, to address use cases requiring significant resource usage.

The use of the phrase "significant resource usage" is also imprecise. Significant resource usage depends on the amount of resources an infrastructure has and is also dependent on the technological advancement at that point in time. A computing centre with a cluster of 100 cores will see 50 cores as a significant resource usage while a computing centre with a cluster of 10,000 cores would probably interpret it differently. A simulation application that uses tens of thousands of cores was a major achievement a decade ago but is now simply a well-scaled application due to the technological advances in HPC and algorithmic optimisation.

In our view, the requirements of urgent computing from Trebon are found to contain imprecise terms that require clarifications, inaccurate representation on the characteristics

of urgent computing and requirements that are too specific to SPRUCE for general application. The shortcomings in the three requirements are addressed in Chapter 3 with a refined and extended definition of urgent computing.

2.1.3 Urgent computing use cases and challenges

There are a variety of use cases worldwide that can benefit from urgent computing to compute high fidelity forecasts. Depending on the use case and the classes of computing resources used, the challenges can differ and thus a diverse set of solutions are proposed. Table 2.1 shows a selection of use cases and challenges, and the solutions found in a research [7] over the Internet that are suggested or implemented. It includes use cases from different origins to demonstrate a global coverage.

Use case/Challenge	**Origin**	**Regular computation**	**Solution/Suggested solutions**
Flash flood [40]	Italy (Genoa)	No	HPC resources
Weather forecast on PLX [41]	Italy	Yes	Dedicated HPC resource
Earthquake early warning system [42]	Japan	Yes	Dedicated resources
Tracking real-time storms in project LEAD [43]	USA	No	Elevated priority (shared HPC resources)
Storm surge computation [28]	USA	Yes	Dedicated HPC resource
Forest fire propagation [44]	Spain	No	Generic algorithm (possible with both dedicated and shared resources)
Using smartphones on cloud [39]	USA	No	Mobile phone and/or cloud resources
Implementing computing on HPC system [45]	USA	No	Innovative allocation policies and scheduling strategies (shared resources)
High-level knowledge based structures for simulations [38]	Russia	Not applicable	Build decision support systems (shared resources)
Interactive Workflow Infrastructure [46]	Russia	No	Reactive programming paradigm with interactive workflow model based on blocks (Shared resources)
Resource management for flood decision support [47]	Poland	No	Timely data acquisition and delivery with holistic approach (cloud resource)

Table 2.1: Urgent Use Cases, Challenges and Solutions

Three classes of resources are utilised by the above use cases and challenges and are shown as follows:

- Public (Shared) resources
- Dedicated resources
- On-demand resources

Table 2.1 shows that the dedicated resources (fourth column) are chosen when regular urgent computations (third column), frequently occurring use cases, are required. Public and on-demand (cloud) resources are used when urgent computations are required less regularly (rarely occurring use cases). Public resources in the form of e-Infrastructures are particularly useful for urgent computing. The term e-Infrastructure refers to "a new way of conducting scientific research by the creation of a new environment for academic

and industrial research in which virtual communities have shared access to unique or distributed scientific facilities (including data, instruments, computing and communications), regardless of their type and location in the world." [48]. In Europe, examples of such e-Infrastructures are PRACE and EGI. PRACE provides leading HPC resources for European scientists while EGI coordinates and manages a large number of distributed Grid and Cloud resources for various research communities. In the US, XSEDE provides an e-Infrastructure supporting more than 2000 projects [49]. More details on different classes of computing resources are shared in Section 3.5.

It is also worthwhile to note that among the use cases shown in Table 2.1, only one use case, forest fire propagation in the sixth row, proposed an algorithmic solution. This is not reflective of the importance of algorithmic solutions in urgent computing. The event domain scientists with whom we work clarified that they have only in the recent decades begin to actively use HPCs for their research works. The exciting possibility of computing at zero hour has only been newly introduced to them. Thus, commonly used research codes, e.g. SPECFEM3D Cartesian[5], are still unsuitable for urgent computing in terms of performance and functionalities. Urgent computing will require fast performance to meet the stipulated deadline. Required urgent computing functionalities, e.g. fault tolerant and adaptive ability to handle evolving zero hour data, will also be useful to improve the reliability and also the accuracy of forecasts.

In principle, there are already algorithmic research activities that are beneficial to urgent computing. In Japan, a real-time inundation forecast system for tsunami [50] was developed. The forecasts can be computed under eight minutes [51, p.26] by scaling up the TUNAMI (Tohoku University's Numerical Analysis Model for Investigating Tsunami) code with a staggered leap-frog 2-D finite difference numerical scheme [52] on the targeted computing resources. Fault tolerant algorithms that are able to recover from faults are particularly useful to avoid compromising the timing constraints of urgent computations in event of faults. Possible strategies include the combination use of chaotic relaxation and meshless methods [53], and sparse grid combination technique [54]. When applied appropriately, these methods can potentially allow a computation to continue without compromising the result when a certain number of tasks/cores, limited by a threshold, fail during runtime. Adaptive algorithms that handle evolving data can also contribute to improving the convergence and the accuracy of urgent computing by incorporating new/updated data in running computations. Numerous adaptive algorithms have been proposed for evolving data, e.g. performing only a partial update of the coefficients in a given iteration while using the nonlinear Volterra filter [55] and the use of CVFDT (Very Fast Decision Tree learner from continuously-changing data streams) algorithm to perform data mining on time-changing data streams [56]. Such algorithms can potentially assist urgent computations to arrive at more reliable and accurate results and/or for the numerical solutions to converge faster. However, these advancements have yet to make their way into the event domain science forecast models. More effort is still required in incorporating such algorithmic solutions, which is outside of the scope of this dissertation.

[5]https://geodynamics.org/cig/software/specfem3d/

2.1.4 Urgent Computer Systems

There are currently a number of urgent computing systems, e.g. SPRUCE and UrbanFlood Common Information Space (CIS) [30]. CIS is a part of an early warning system that works on top of the Web Services Business Process Execution Language (WS-BPEL). It is argued that the CIS framework is potentially applicable to other early warning systems, which will not have similar workflows, but could potentially benefit by reusing sub-workflows that have a high repeatability rate. Since the underlying computing resources are cloud resources, heterogeneity is not an issue as the urgent computing environment, e.g. operating system, available libraries, and software applications and their versions, can be fixed by having virtual images of the urgent computer system and the targeted computing resources. This differs from our proposed approach, which requires a wide array of computing resources such that the ensembles of forecasts can be completed simultaneously within the time constraint. E-Infrastructures, which offer a wide array of heterogeneous resources, are our targeted class of resources. Due to the heterogeneity of such computing resources and their environments, the "sub-workflow", i.e. tasks/subtasks, are environment/resource specific and non-repeatable.

An interactive workflow model [46] based on a reactive programming model was also proposed. The aim is to build a workflow-driven infrastructure to enable interactive capabilities in soft real-time. The proposed model is a part of a bigger platform, CLoud Applications VIRtual Environment (CLAVIRE) [57], to provide the complete urgent system. It was assumed that the most crucial criterion, deadline, can be met by making predictions on the calculation time for the specific problem statement. Computing resources are assumed to be limited but sufficient. The approach shares some similarities with our proposed solution. Both aim to provide flexibility to cater for inevitable modifications to the workflow or work process. Our proposed solution is additionally able to fulfil the requirement to support ubiquitous devices during chaotic times when a disaster is impending or has struck. Practical security concerns, which have been a major road block to many proposed solutions are also taken into consideration in our solution.

2.2 Resource Allocation

In this section, resource allocation strategies in urgent computing and resource allocation heuristics are discussed. These researches are related to the proposed research allocation heuristics in Chapter 5.

2.2.1 Resource Allocation Strategies in Urgent Computing

Resource allocation within SPRUCE was based on an opportunistic approach [28] and/or on elevated-priority policies [12], i.e. next-to-run, preemption, etc. In spite of the efficiency of preemption, it was not commonly adopted due the policies of providers. Next-to-run and oportunistic scheduling were more common but were reported to cause delays [28] that

were detrimental to meeting the deadline.

In another related work [58], an attempt to study the impact of urgent computing on resource management policies, schedules and resource utilisation on three resources was carried out. Four different resource allocation strategies are proposed:

(i) Load-balancing via the minimum queue length

(ii) Dedicated (one) resource provider for urgent jobs

(iii) Combine the above two. If no dedicated resource is available, choose the resource with the minimum queue length

(iv) Minimum completion time policy

In both works, a single job/forecast, as opposed to an ensemble of forecasts was examined. Both focused mainly on the computing resource challenges and not the common limitations of the event domain sciences, which were insufficient for practical reasons. To effectively allocate a set of heterogeneous resources for an ensemble greatly increases the challenge since different forecasts and resources will have different execution time performances.

In another urgent computing related work for early warning system (EWS), a hybrid scheduling algorithm for urgent workflows [59] based on traditional and meta-heuristic approaches within cloud resources is developed. The EWS hybrid scheduling algorithm manages multiple forecasts but since they are not a part of an ensemble, the requirements are different.

2.2.2 Resource Allocation Heuristics

The decision to select an optimal resource allocation is typically either P-complete [60] or NP-complete [61] as it involves a number of objectives, which cannot be simultaneously optimal. Consequently, heuristic algorithmic approaches are prevalent to deal with such decision problems.

Among the heuristic approaches for resource allocation, it can be further categorised into those that optimise a single resource allocation [62][63] and those that manage distributed resource allocation [64]. Distributed resource allocation, which we are concerned with, includes numerous robust resource allocation heuristics [60][65] where deadline as a constraint is proposed. Most works focus on how to efficiently fit jobs/tasks onto a limited set of resources, under the assumption that there are insufficient resources to execute all jobs simultaneously. This differs from our requirement (refer to Chapter 5), which requires all jobs to execute simultaneously to meet the stipulated deadline.

With rising energy costs, energy- and performance-aware resource allocations are gaining importance to save energy and costs. At first glance, the cost of energy appears trivial when compared to the potential losses of urgent events. However, studies [66][67] have shown that the energy constraint faced by resource providers is very real. Energy consumption has increased over the years and in 2010, between 1.1 to 1.5% of global electricity

use is believed to be from data/HPC centres. At LRZ, approximately a third of its funding is used to foot the power bill. Intel Turbo Boost technology[6], which uses the highest possible frequency achievable at the expense of energy, is unsupported (disabled on purpose) during day-to-day operations. The existing power capacity, 10 MW, of LRZ does not offer the possibility to host the current (November 2015) top HPC resource, Tianhe-2, in the Top500 list[7], which requires 17.8 MW of electrical power to have a peak performance of 54.9 PFlop/s. The upcoming Exascale systems [68][69] are expected to require even more electric power. As the budget dedicated to urgent computing is expected to be fixed and limited, the more energy efficient an resource allocation is, the more urgent computations can be supported.

There is thus a need to minimise the energy usage such that the resource providers can readily support urgent computing without adding unnecessary burden and strain to their existing power infrastructures. A variety of heuristics [70] for scheduling of tasks is proposed. Most approaches leverage on single resource, i.e. a single process platform [62] or a multi-core platform [71]. The main concern is to find an optimal compromise between power and the defined performance constraints, e.g. deadline, timing, job completion time, quality of service. The most common technique used is to manipulate the energy consumed by adjusting the voltage or frequency of nodes/cores/processes of resources within the application source codes or via system specific kernels/software. In our solution, the ETS values are assumed to be known. The heuristic algorithms are independent of the actual technique used.

In the works [72][73], heuristic/algorithmic approaches to reliably allocate tasks on distributed systems and heterogeneous clusters are proposed respectively. Both attempt to allocate the tasks, which require interprocessor communication, within a single job to various processors of a system. This differs from our requirement where the targeted set of resources is made up of independent dynamic heterogeneous HPC systems, e.g. resources on e-Infrastructure like PRACE. Multiple independent jobs can thus be scheduled simultaneously. This is also the first heuristic algorithmic attempt in urgent computing to optimise the following multi-objectives, robustness, site and resource reliability, and ETS for ensembles of forecasts.

2.3 Ensemble of Forecasts

There are many urgent computing events, e.g. storms and wildfires, which are very localised in space and in time, making them hard to predict. In such cases, there is no single forecast model or set of conditions, i.e. physical parameters, initial and boundary conditions, which can guarantee a high-fidelity forecast. The inherent limitations of most observational data and forecast models, forecast uncertainties [74][75] are generally unavoidable even at zero hour. Main limitations in observational data can be attributed to

[6]http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html

[7]http://top500.org/list/2015/11/

data errors, missing data and/or conflicting observed data while core limitations in forecast models are related to the incomplete understanding and modelling of the underlying science, e.g. the physics, and the simplified numerical representation of complex processes. Consequently, stochastic [5] as opposed to deterministic forecast methods are increasingly prevalent. Ensemble forecasting [6], which can include the simulations of a number of numerical models, and/or perturbations in initial conditions and/or physical parameters, provides the basis for stochastic predictions.

Currently, there are many operational weather prediction services worldwide that take advantage of ensembles of forecasts. The list below illustrates a selected set of these systems and centres, and there are many others.

- European Commission's Flood Awareness Systems [76] [77]
- European Centre for Medium-Range Weather Forecasts (ECMWF) [78]
- Germany's Deutscher Wetterdienst (DWD) [79]
- The US's National Centers for Environmental Prediction (NCEP) [80]
- China Meteorological Administration (CMA) [81]
- Australia's Bureau of Meteorology [82]
- Japan Meteorological Agency (JMA) [83]

Consequently, our proposed urgent computing solution will support ensembles of forecasts to improve the fidelity of the computed predictions. The number of forecasts required per ensemble varies and is dependent on the use case.

2.3.1 Flooding

Flooding is an example of an urgent event that can greatly leverage on urgent computing. According to the findings of an ensemble flood risk assessment study [84] that was carried out by the European Commission (EC) Joint Research Centre (JRC), the frequency and damage of floods in Europe are expected to increase sharply over the next decades. Table 2.2 shows the current (Year 2015) and projected figures (Year 2050 and Year 2080) on population affected and economic loss. A range is presented for the projected figures to provide room for variation in future economic growth. Both the number of population affected and economic loss are expected to increase. By 2080 (refer to the last row in Table 2.2), the number of affected population is expected increase by 2- to 4-fold as compared to now while the cost of damages is expected to increase by 5- to 18-fold. In Germany alone, 170,000 to 323,700 of the population are expected to be affected by floods with a damage of 5.3 to 33.9 billion euros in 2080.

Consequently, EC developed services like the European Flood Awareness System (EFAS) [76] to provide early warnings from three to ten days by monitoring and forecasting floods across Europe. Naturally, the flood threat is not restricted to Europe. Thus,

Year	Population Affected	Economic Loss
2015	216,000	5.3 B€
2050	500,000 - 640,000	20 - 40 B€
2080	540,000 - 950,000	30 - 100 B€

Table 2.2: Result of Flood Risk Assessment Study for Year 2015, 2050 and 2080

the EC also developed a Global Flood Detection System (GFDS) [85] and Global Flood Awareness System (GIoFAS) [77] by leveraging on satellite-based microwave observational data to produce near real-time maps and alerts for major floods, and an integrated hydro-meteorological forecasting chain and a monitoring system to forecast flood events with a lead time of up to 30 days respectively.

One form of flood, flash flood [86][87][88][89][90], receives additional attention. It is arguably the most dangerous type of floods as it can form swiftly due to high or extremely high rainfall rates with little or no prior warning and with devastating consequences especially in urban regions. Consequently, JRC carried out additional studies [91][92] to understand the feasibility of increasing the lead time using probabilistic methodologies to enable early warning. However, due to the limited spatial scope of flash floods, Global Climate Models (GCMs), which are in the scale of 15-25 km have difficulties in detecting them. In such cases, Limited Area Models (LAMs) operating at cloud permitting grid spacing (5-1 km) had to be used [86][88].

One example of LAMs is the THORPEX[8] (a World Weather Research Programme) Interactive Grand Global Ensemble-Limited Area Model (TIGGE-LAM)[9]. It is an extension of the TIGGE archive to include regional weather forecasts from LAM ensembles with grid spacing of 12 and 2 km resolution to provide detailed information up to a few days ahead to complement the larger-scale information provided by the global data in the established TIGGE archive.

A flash flood that occurred on 9 October 2014 in Genoa, Italy within a spatial scope of less than 15 km^2 is an example requiring a LAM. This particular event will be used to demonstrate the proposed heuristics in Section 6.2. Both EFAS and GIoFAS systems do not have a record of this flash flood, i.e. the flood was not detected by either system. The national weather service in Italy only issued a heavy rain and storm weather forecast. No flood warning was issued. Unfortunately, there were one death and an estimated damage of 303 million USD based on data from the International Disaster Database (EM-DAT)[10].

Flash floods [87] are thus events that can greatly leverage on urgent computing, due to their particularly short lead times. There are also a number of flood related urgent computing research work [47][30] for decision support and early warning. Support for ensembles of forecasts are not mentioned in these approaches. There is no known urgent computing research work on enabling ensembles of forecasts.

[8] http://www.wmo.int/pages/prog/arep/wwrp/new/thorpex_new.html
[9] https://software.ecmwf.int/wiki/display/TIGL/Home
[10] http://emdat.be

Chapter 3

Urgent Computing Definition

Urgent computing requires computations to commence in short order and complete before a stipulated deadline so as to support mitigation activities in preparation, response and recovery from an event that requires immediate attention. In order to understand and tackle its unique requirements and challenges, a comprehensive definition is required. In this chapter, a refined and extended urgent computing definition is shared. We begin by gathering the requirements, both functional and non-functional, in Section 3.1 to understand what needs to be done. This is followed by the identification of the characteristics, pre-computation and post-computation, in Section 3.2 to help identify urgent computations. In the next two sections, deadline and cost of urgent computing are shared. Finally, the different classes of computing resource, public, dedicated and on-demand, which influence the cost of urgent computations, are classified in Section 3.5. A set of policy recommendations that are required to successfully realise urgent computing on our targeted resource class, public resources, is also shared.

3.1 Requirements

In this section, the requirements of urgent computing are presented. Figure 3.1 provides an overview of the functional and non-functional requirements. The functional requirements consist of data collection, human machine interface (HMI) and data evaluation, which are discussed in more detail in Section 3.1.1. Non-functional requirements consist of robustness, reliability, maintainability, availability and security, which are shared in Section 3.1.2. The functional requirements are made up of practical functionalities that are required to realise urgent computing. These requirements are rather generic and are in fact also applicable to real-time systems (RTS) and non-urgent computing systems, e.g. CDMC systems. The non-functional requirements are more crucial in highlighting the unique and more

stringent requirements of urgent computing. These requirements are the de facto qualities of a well-design system but are typically relegated to a lower priority or ignored since the cost of fulfilling them are perceived to outweigh the benefits. However, in the case of urgent computing, they are imperative requirements. SPRUCE was not concerned with this aspect of urgent computing and thus there was no mention of functional and non-functional requirements in their research works. We however believe it to be very important and have thus defined them in this dissertation.

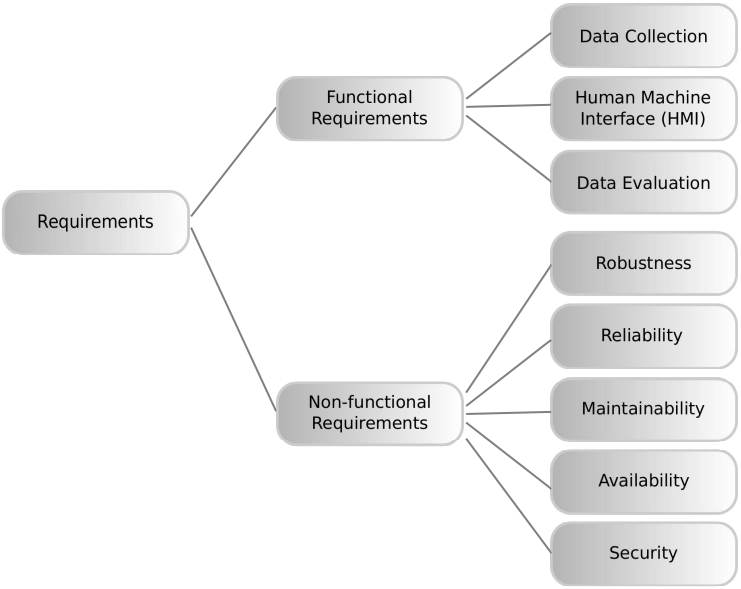


Figure 3.1: Tree Diagram of Urgent Computing Requirements

3.1.1 Functional Requirements

The functional requirements of urgent computing are gathered by analysing the functional requirements of the very similar paradigm, real-time computing [2], and evaluating the requirements of the urgent use cases, e.g. [40][27][42][45]. It is thus comprised of data collection requirements and human machine interface (HMI) requirements. Additionally, to facilitate decision-making, data evaluation functionality must also be included. These three functionalities are elaborated as follows:

Data Collection

Data collection is arguably one of the most fundamental requirement of urgent computing. Data is essential to determine if an urgent computation should be triggered, and where and how it should be initiated. Urgent event data, i.e. observational data, is the most crucial input to forecast models and is typically only available in short order before/when the event strikes. Computing resource data is indispensable to determine the available and most suited set of resources for urgent computations within the stringent time constraints.

Data can be collected via two mechanisms, time-triggered and event-triggered. Time-triggered mechanism, e.g. system/service monitoring, is the most common method due to its simplicity to implement. However, depending on the scheduled time-interval, a delay, equivalent to the prescribed time-interval, can ensue. For example, if a failure occurred right after the time-triggered monitoring is performed, the failure will not be recognised until the next time-triggered monitoring occurs. Time-triggered data collection is an important mechanism to detect abnormalities in a system in advance to facilitate both the efficient resolution and avoidance of failing components during an urgent event.

Event-triggered mechanism represents a "real-time" method where observed irregularities/abnormalities and non-regular activities trigger appropriate activities. Examples of such triggers are an irregularity in observational data that can indicate the onset of an urgent event and an update of the operating system of a computing resource that can limit its availability and/or reliability. Event-triggered mechanism is particularly useful for non-deterministic triggers, which cannot be easily designed into time-triggered mechanism. The use of sensors is a method to provide automatic event triggers where time delay can potentially be minimised to near zero.

Human Machine Interface

Human machine interface represents the interface between an urgent system and its operators. If irregularities/abnormalities are observed in the data collection activity that indicate an impending or occurring urgent event, it enables the operators to trigger urgent computations on the urgent system. Such an interface is required be it that an urgent system is non-autonomous, partially autonomous or fully autonomous. In the case where the system is fully autonomous, an HMI interface is required to override the system when necessary.

Unlike RTC, this interface is not as safety-critical since the decision-making and coordination system is mostly a separate component (refer to Section 4.1). However, an incorrectly implemented HMI interface can still lead to an interruption of services on public shared resources or leasing of substantial cloud resources that can be financially costly.

Due to the chaotic environment that frequently accompanies an impending or occurring urgent event, there are several HMI design criteria that are essential. The interface has to be simple and easy to interpret even in chaotic circumstances. Since the operator(s) might not be in the vicinity or have difficulties travelling to their workplace in chaotic times, the integration of the interface on ubiquitous devices is thus crucial. Finally, the flexibility to update the forecast models via the interface must be included. This is to accommodate possible modifications to the configurations of forecast models or the addition of new forecast models. Modifications are typically performed to refine the forecast models as a result of additional insights from zero hour data.

Data Evaluation

Data evaluation is another core functionality of urgent computing. The urgent products from computations have to be evaluated to provide authoritative information on the urgent event for educated decision-making and mitigation activities. If ensemble of forecasts are involved, this function will have to leverage on forecast evaluation/prediction models, e.g. forecast probabilities [93] and bayesian forecast evaluation [94]. Depending on the event and its physical environment, there will be some forecast models with a higher forecast confidence when compared to others. Consequently, there is no single evaluation/prediction model that can be applicable across the board. Machine learning methods [95][96] can potentially help in automating this and/or to provide additional insights as to which models are most effective under which set of conditions.

Since decisions and mitigation activities are tightly coupled with this function, mistakes, e.g. the failure to issue a warning or the publication of an incorrect forecast, can have catastrophic consequences. Instead of the desired loss mitigation, it could potentially cause more casualties as in the case of the L'Aquila earthquake in 2009[1] where seven scientists shared an incorrect prediction and consequently caused avoidable deaths. The Italian court thus gave them an initial conviction for manslaughter. This particular event took a mere twenty seconds to cause 297 deaths, more than a thousand injured and around 70,000 people to become homeless[2]. Inaccurate/incorrect predictions have far-reaching effects, in particular a loss of public confidence, which can result in civilians ignoring crucial warnings and thus cause more casualties. It is particularly challenging to rectify a loss of public confidence. It is thus imperative to ensure that computed data is correctly and accurately evaluated.

To facilitate the evaluation, the forecast results, i.e. urgent products, have to be processed and reorganised in a useful manner for easy analysis such that appropriate decisions and mitigation activities can be taken before the stipulated deadline. Data analytic and scientific visualisation are important mechanisms to efficiently support this function.

3.1.2 Non-functional Requirements

Non-functional requirements relate to the success of urgent computing. They are the underlying criteria determining the effectiveness and efficiency of an urgent system. These are also the requirements that set urgent systems apart from many other computing systems that do not have stringent non-deterministic timing constraints. The non-functional requirements are gathered by analysing the non-functional requirements of real-time computing [2], evaluating the urgent computing constraints, in particular the time constraints, and the requirements and policies of computing resources.

[1] http://www.foxnews.com/science/2012/10/22/italian-court-convicts-7-scientists-for-failing-to-predict-earthquake.html

[2] http://www.360cities.net/image/terremoto-aquila-casa-dello-studente

Robustness

Robustness is defined as the tolerance of a computing system to perform faithfully in event of perturbation(s). In the case of urgent computing, it denotes the tolerance of the selected computing resources, services and tools to support the completion of required forecast simulations within a stipulated deadline, in spite of possible (minor) changes in the computing system. It can be seen as the most crucial attribute that dictates the success of an urgent system by ensuring that timing constraints are fulfilled in case of failures. Robustness can be improved by designing tolerance into the system.

Reliability

Reliability is the ability of a computing system to perform the required functions correctly within a specified time period. Unlike robustness, reliability does not manage perturbations in the computing system but is mainly concerned with the trustworthiness of a system during normal, i.e. expected, operation. In the case of urgent computing, the reliability of an urgent system is the probability that a specified urgent service is carried out correctly before an instance in time, the urgent deadline. Reliability of urgent services is dependent on the selection of reliable computing resources, the correctness and accuracy of the simulated forecast results and derived decisions, etc. Reliability in computing resources is dependent on the condition that the resources are operational, i.e. available, at the time when they are initiated. Reliability can be improved by provisioning alternative failover components and should ideally be in different localities, i.e multiple resource providers from different spatial locations. In event of a natural disaster, there could be a chance that a particular resource in the affected region becomes unavailable. In the aftermaths of the March 2011 tsunami in Japan, the damage to the power grid and rolling blackouts rendered their computing resources non-operational [97]. A cloud outage caused by an electrical storm at the Amazon Web Services resulted in the shutdown of hardware in an entire region [98].

Reliability in simulated results and the derived decisions is crucial since it is tied to the success of urgent computing. It is dependent on the forecast models, the adopted prediction models and frequently the experience of decision makers in interpreting the results. Incorrect and/or inaccurate forecasts can erode the confidence of civilians and have far-reaching effects that can ultimately lead to a loss of human lives if accurate warnings are ignored. Thus it is critical to stress the importance of reliability of forecast models and the decision-making process even if it is outside the scope of this dissertation.

Maintainability

Maintainability of an urgent system is the probability that the full system is recovered within a specified time interval after a failure or change. Similar to RTS, there is a fundamental conflict in reliability and maintainability [2]. The need to have failover components for reliability increases the complexity of the system and thus the maintainability. One

factor contributing to the failure of SPRUCE can be attributed to the failure to achieve maintainability upon a change in a key tool, Globus (refer to Section 2.1).

Availability

Availability of an urgent system is the probability that an operational system, albeit not fully, is available over the planned operational time interval of the system. It can be measured by the fraction of time that the urgent system is ready to provide the urgent services [2]. Availability can be used to represent the reliability of a computing resource as shown in Section 5.2.2. The choice of higher availability computing resources will improve the overall reliability of an urgent system. The availability of services and tools running on the selected computing resource that is required to support urgent computations can also influence on the availability of an urgent system. If these services and tools are unavailable, the computing resource is correspondingly unavailable. This could potentially reduce the availability and reliability of the urgent system.

Security

Security of an urgent system pertains to the "trustworthiness" of the system. It is a key attribute that influences the integrity of the system and its associated components. In order to ensure that the results and time constraints are not adversely influenced due to security breaches, designing strong and comprehensive security mechanisms into the urgent system is crucial. Security concerns can result in resource providers, in particular the public ones that we are targeting, to limit or refuse to support urgent computations. It is thus important to ensure sound security concepts are implemented in urgent systems.

3.2 Characteristics

There are two observed groups of characteristics, pre- and post-computation, which can be used to differentiate urgent computations from related computing paradigms (refer to Section 2.1.1) computations. The definitions of the two groups and their respective characteristics are in the following subsections. All pre- and post-computation characteristics have to be fulfilled for a computation to be considered as an urgent computation.

3.2.1 Pre-computation Characteristics

Pre-computation characteristics describe the conditions surrounding an event that indicates an urgent computation(s) is required.

An overview of the five identified pre-computation characteristics is shown in Figure 3.2. They include (i) potential extensive loss, (ii) data unavailability until zero hour, (iii) expect improved accuracy results, (iv) computations to start immediately or asap and (v) errors from estimated time constraints. The details of these characteristics are shown in the same order as follows:

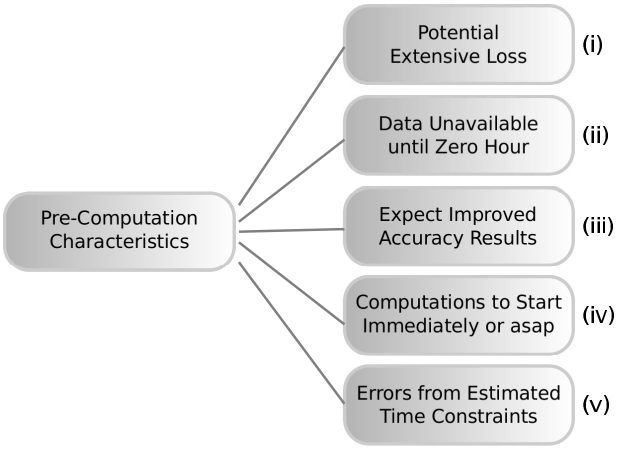


Figure 3.2: Overview of Pre-Computation Characteristics

(i) The urgent event is expected to potentially result in extensive loss to multiple stakeholders and should include at least one government/government-authorised agency.

(ii) Data required to set up the initial condition of an urgent computation is unavailable until zero hour, i.e. shortly before/when/after the urgent event is observed.

(iii) Simulation results with improved accuracy are expected by using zero hour data.

(iv) Computations must commence immediately or as soon as possible in order to meet the deadline and thus computing resources must be available on short notice.

(v) Provided time constraints, i.e. the deadlines, are estimates and the error can be in seconds, minutes, hours or even days.

One important characteristic of urgent computing is the (i) potential extensive loss, typically in the range of multiple millions to billions of euros. Table 3.1 shows the estimated economic loss in tens to hundreds of billions of euros (last column) of four severe but different disasters (first column) worldwide (third column) in recent years (second column). Typically, it will take a number of years before the full extent of the loss can be collected. Multiple stakeholders, including the government agencies, are expected to be affected. When compared to such losses, the cost of urgent computations (refer to Section 3.4) is typically insignificant. The use of the term "extensive loss" raises the question of how extensive a loss must be to warrant urgent computations. A common question posed is "Is a single human life a sufficiently extensive loss?". In the US, the Environment Protection Agency set the value of a human life at 9.1 million US dollars in 2011 while the Food and Drug Administration declared life was worth 7.9 million US dollars in 2010[3]. It is thus a philosophical and political question that is outside the scope of this dissertation. In principle, the decision lies in the hands of the decision makers.

[3] http://www.nytimes.com/2011/02/17/business/economy/17regulation.html?pagewanted=all&_r=1

Table 3.1: Economic Loss of Some of the Most Severe Disasters in Recent Years

Disasters	Year	Country	Estimated Loss (€)
Flood [99]	2013	Germany	15 billion
Tōhoku Earthquake/Tsunami [100] [101]	2011	Japan	273 billion
Deepwater Horizon Oil Spill [34]	2010	Gulf of Mexico	53 - 88 billion
Hurricane Katrina [102]	2005	United States	88 billion

The main motivation to perform urgent computations is the expectation of improved accuracy in the simulation results by using "live" data at zero hour. The next two characteristics, (ii) and (iii), help to filter out use cases that do not require zero hour data for the computation and/or will not benefit with an improved accuracy. An example from CDMC is the Forecast Propagation Database for Tsunami [103], a non urgent computing solution, where an exhaustive set of possible scenarios are pre-simulated and the results kept. Upon an impending tsunami, the best match simulation from the database is used to predict the event. This solution is currently one of the most valuable solution for tsunami early warning system. The scientist we work with explain that although it does not have the expected accuracy that one can possibly achieve with urgent computing, the inherent uncertainties in locating the seismic source and understanding the geological properties make it less significant to use approximated data as opposed to zero hour data.

Characteristic (iv) shares that to satisfy the time constraint, i.e. deadline, the computations must commence swiftly upon getting the required data. Computation resources should thus be available in short notice correspondingly. If the computations do not have to commence in short order, it is not an urgent computation.

Finally characteristic (v) shows that as opposed to most real-time computing events where time constraints/deadlines are known, i.e. deterministic, the time constraints of urgent computing are often unknown and are a part of the targeted computation results. As such, they are typically estimates and errors are expected. In the best case scenario, the errors can be in the range of seconds or minutes but they can be also in the range of hours and even days in the worst case scenarios. This unpredictable characteristic makes the realisation of urgent computing more challenging than hard real-time computing.

3.2.2 Post-computation Characteristics

Post-computation characteristics describe the conditions surrounding an urgent event after the computation(s) are completed.

An overview of the four identified post-computation characteristics are shown in Figure 3.3. They include (i) computation completes before deadline, (ii) mitigation activities to start immediately, (iii) urgent products used for mitigation and (iv) direct loss expected. The details of these characteristics are shown in the same order as follows:

(i) An urgent computation must generate urgent product(s) before a stipulated deadline for it to be useful.

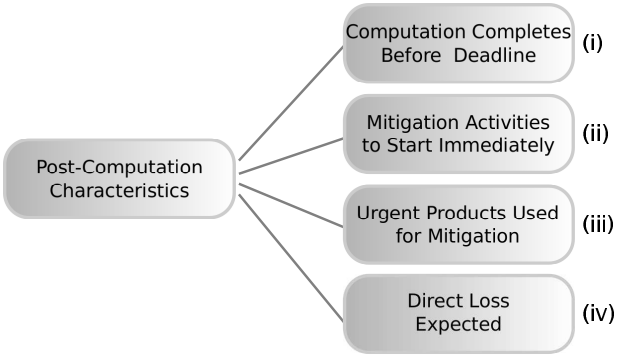


Figure 3.3: Overview of Post-Computation Characteristics

(ii) Decision and coordination activities commence or recommence immediately upon the availability of the urgent product(s).

(iii) The decision makers use the urgent product(s) to make informed decision(s) to mitigate the loss of the event.

(iv) Direct loss of the event can be expected though mitigated upon meeting the stipulated deadline, and after making and carrying out the informed decisions.

Meeting the stipulated deadline, (i), in urgent computing is crucial to the success of system. Lateness may imply that no mitigation effort can be carried out. Meeting the deadline thus constitutes one of the most crucial and yet fundamental requirements to the post-computation characteristics.

Decision and coordination activities, (ii), have to immediately commence upon the availability of the urgent product(s) for urgent computing to make sense. If the decision and coordination activities can wait then it does not require urgent computations. The importance of meeting the stipulated deadline originates from the need for decision-making such that mitigation activities can be carried out. Naturally, the more lead time there is for mitigation activities, the better can be the plan and the overall reduction in severity of the loss (refer to Figure 3.5).

Characteristic (iii) shows an important difference, whether direct loss of the event will be incurred upon meeting a deadline, between urgent computing and real-time computing. A RTS is designed such that in a deterministic set of scenarios, if a deadline is met, no direct loss/cost is incurred. However, in the case of urgent system, even in an ideal prescribed scenario, direct loss can still be expected upon meeting the deadline. The main aim is to mitigate the loss since complete avoidance is likely to be impossible.

Finally characteristic (iv) shares that urgent product(s) should provide the necessary information to support the decision makers in making informed decisions to reduce the severity, i.e. the loss, of the event. If urgent products are unnecessary for the decision-making process or mitigation effort, then the urgent computations are correspondingly unnecessary.

3.3 Deadline

A *deadline* is a point in time where a predefined task(s) has to be completed. The time constraints or "in time" requirements of urgent computing can be expressed in terms of deadlines. A deadline thus helps to define the minimum lead time required for decisions to be made and carried out, which in turn contributes to a positive loss mitigation.

To manage the unpredictable timing characteristic, a refined definition of urgent system deadlines is required. Urgent computing deadlines can be either firm or hard. If the urgent products are helpful, e.g. not the only source of information, for the decision-making, the deadlines are firm. If the urgent products are crucial, e.g. the only source of information, for the decision-making, the deadlines are hard.

The deadlines of urgent systems are defined as such:

- **Firm deadline**: A deadline is firm if not meeting it renders the result useless but the consequences are not very severe.
  - ***Deterministic* firm deadline**: A deadline is deterministic firm if the deadline is predictable and not meeting it renders the result useless but the consequences are not very severe.
  - ***Non-deterministic* firm deadline**: A deadline is non-deterministic firm if the deadline is unpredictable and not meeting it renders the result useless but the consequences are not very severe.
- **Hard deadline**: A deadline is hard if the consequences of not meeting it can be catastrophic.
  - ***Deterministic* hard deadline**: A deadline is deterministic hard if the deadline is predictable and the consequences of not meeting it can be catastrophic.
  - ***Non-deterministic* hard deadline**: A deadline is non-deterministic hard if the deadline is unpredictable and the consequences of not meeting it can be catastrophic.

Figure 3.4 shows an overview of the common characteristics shared among the deadline definitions. In the first and second columns, the characteristics of deterministic deadlines and non-deterministic deadlines are shown respectively. In the first and second rows, the firm and hard characteristics of the deadlines are elaborated and highlighted with corresponding pink and gray hexagons respectively. In the first and second columns, the deterministic and non-deterministic characteristics are illustrated with corresponding yellow and gray hexagrams respectively. The characteristic of urgent computing, which is common among all urgent deadlines, is shown with blue crescents. For example, the top left rectangle contains the characteristics of a deterministic firm deadline.

In summary, deterministic deadlines are predictable while non-deterministic ones are unpredictable. Firm deadlines imply not very severe consequences while hard deadline

indicates possible catastrophic consequences. All urgent deadlines share the common characteristics where results become useless if deadlines are missed.

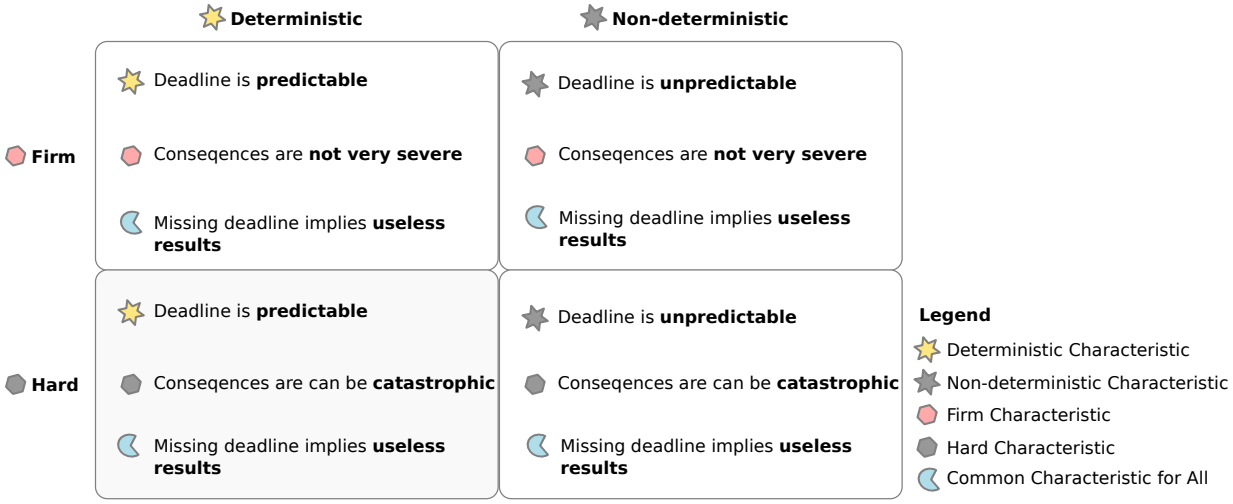


Figure 3.4: Overview of Common Characteristics among the Deadlines

One crucial difference between urgent deadlines and real-time deadlines is the expectation of consequences even under ideal conditions where an urgent system is functioning as expected and the stipulated deadline is met. This is as opposed to a hard RTS where meeting the deadline under pre-defined scenarios means avoiding catastrophe consequences. Urgent systems are frequently only attempting to mitigate the consequences such that it will not be catastrophic. For instance, the early earthquake warning system is only useful to those who are of a certain distance from the epicentre of an earthquake. The warning is useless to the civilians living close to the epicentre. Additionally even though deterministic firm and hard deadlines of urgent computing are synonymous with the firm and hard deadlines of RTS respectively, non-deterministic deadlines are unique and more common to urgent computing. This further differentiates it from real-time computing. The non-deterministic deadlines of urgent computing are the deadlines there are highlighted with the * symbol in Figure 2.2.

Due to the lack of information until zero hour and the dynamic nature of many urgent events, determining a deadline is frequently impossible. Such deadlines can only be estimated and the models used are typically best estimations. A significant inherited error is thus expected. The non-deterministic characteristic invalidates many well-proven solutions of hard RTS since it is impossible to design a system to exhaustively handle dynamic and unpredictable scenarios.

Three deadline variables are thus defined to express the characteristics of urgent deadlines as follows:

- t_{ideal} illustrates a point in time where the required minimum lead time for maximum mitigation activities, i.e. minimum loss, can be achieved. After this point in time, the loss is expected to increase exponentially.

- $t_{deadline}$ illustrates a point in time where only the minimum mitigation activities, e.g. evacuate civilians, defined by the decision makers are possible.
- $t_{terminus}$ illustrates a point in time where there is no longer any lead time for any mitigation activities, i.e. maximum loss.

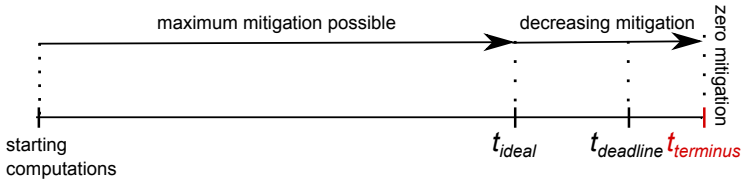


Figure 3.5: Estimated Deadlines

Figure 3.5 illustrates the relationship among the three deadline variables. Maximum mitigation activities to minimise loss are possible if computations are completed before t_{ideal}. If computations are completed after t_{ideal}, the amount of mitigation activities that can be carried out is increasingly reduced. From the time $t_{terminus}$ and beyond, no mitigation activities are possible. Maximum loss is to be expected. Typically, $t_{deadline}$ is set to be between t_{ideal} and $t_{terminus}$.

3.4 Cost

Cost is an important characteristic of urgent computing to quantify loss. It helps to determine if an urgent computation is worth carrying out. Two costs, *computation cost* and *event cost*, are crucial to urgent computing. Generally, the cost of an urgent computation (C_c) should be less than or equal the cost incurred from an urgent event (C_e), i.e. $C_c \leq C_e$, for a computation to be considered as worthwhile. However, this is only true in the ideal situation where the actual cost of the event is zero as a result of perfect mitigation activities. As we have established earlier, in the case of urgent computing, that is not likely. A more appropriate representation of the computation and event costs is shown in equation (3.4.1). C_{me} refers to the cost of the event after mitigation activities are carried out by relying on the urgent products from the computation.

$$C_c \leq C_e - C_{me} \quad (C_{me} \leq C_e) \tag{3.4.1}$$

The relationship between C_{me} and C_e can be further expressed by providing a mitigation factor, α as shown in equation (3.4.2). α will be in the range zero and one. Zero represents the ideal scenario where there is no event cost due to perfect mitigation activities while one represents the worst case scenario where no mitigation activities can be made to reduce the event cost, i.e. after $t_{terminus}$ (refer to Figure 3.5).

$$C_{me} = \alpha C_e \quad (0 \leq \alpha \leq 1) \tag{3.4.2}$$

α is to be derived e.g. from previous events of similar nature. By combining equations (3.4.1) and (3.4.2), the relationship between C_c and C_e is thus simplified:

$$C_c \leq (1 - \alpha)C_e \quad (0 \leq \alpha \leq 1) \tag{3.4.3}$$

Equation 3.4.3 provides a mechanism to assist decision makers in measuring whether urgent computing is worthwhile for a particular urgent use case or event. Event cost (C_e) can be estimated in advance using information from similar events that have occurred previously. Cost of an event includes both direct and indirect cost. Examples of direct cost are damages in infrastructures and loss of human lives. Indirect cost can include economic losses from a loss of investment and tourism after the event. The exact cost of an event is hard to estimate due to the difficulties in quantifying some losses, e.g. human lives, and the determination of whether a particular indirect cost is a result of the urgent event [104].

3.5 Classes of Computing Resources

Urgent computing is realised on top of three classes of computing resources, public, dedicated and on-demand, as shown earlier in Table 2.1. Choosing an appropriate resource class will influence the computation costs of urgent events. In this section, resource classes are clarified and three criteria for selecting the most appropriate resource class are shared. A comparison between an on-demand resource and a public resource with similar hardware profiles is performed to illustrate the computation cost. Finally, a set of policy recommendations for the public resource class is shared. These recommendations aim to enable public resource providers to readily support urgent computing.

The three classes of computing resources are illustrated in the following list.

(class 1) Public (Shared) Resources - State-owned or research community resources that are freely available and shared among researchers.

(class 2) Dedicated Resources - Resources that are purchased for the specific purpose of a particular use case of urgent computing.

(class 3) On-demand Resources - Pay-as-you-use resources that are typically offered by profit-oriented entities.

These classes of resources can change with the advent of technology. Within each resource class, there are different resource types, e.g. a HPC resource, a serial cluster and a single workstation. Ideally, the resource type is selected by considering the computational requirements, i.e. parallel or serial computation, number of compute cores, memory, wallclock time, etc., and computational performance. Logically, a top-tier HPC resource with expensive interconnects that can perform serial programming, task-farming and parallel computing is probably the ideal resource to own and utilise for any urgent computing use cases. However, this is unrealistic since cost is also one of the deciding factors for the

resource class that will be adopted. In the following subsection, the selection of resources will be elaborated.

3.5.1 Selection of Resource Class

Selecting an appropriate resource class is one of the initial challenges of urgent computing. The choice is typically based on the following three criteria and in this order:

(i) Computation Requirements

(ii) Cost and Frequency of Computations

(iii) Intangible Criteria

The computation requirements will determine the resources that can be utilised. The resource class with the appropriate resources will then be checked for cost. Cost is dependent on the frequency of computations and the corresponding resource class. In event that cost is comparable among the classes, intangible criteria like preference, control, etc., will guide the selection. Generally, the resource class with the least cost while fulfilling the computation requirements should be chosen.

(i) Computation Requirements

Computational requirements determine the resource type and thus the resource class. Urgent use cases that have very modest computing requirements, which can be satisfied by a single workstation, utilise the dedicated resource class. Use cases that have moderate computing requirements and require a cluster of serial or parallel computers might take advantage of on-demand resources. Use cases that require the highest computing power will have to leverage upon HPC resources. As the cost of top-tier HPC resources is extremely high and difficult to operate, it is frequently not owned as a dedicated resource and is more cost effective as a shared public resource. There are currently very few on-demand resource providers that can provide high performance computing resources for parallel computations. If the computation requirements require top tier, e.g. top ten percent, HPC resources that are listed on the TOP500 supercomputing sites[4], the selection option becomes very restricted. In such cases, public resources will triumph over on-demand resources simply due to availability. Towards the end of this section, a comparison between a public HPC resource, SuperMUC, and Amazon's HPC cluster is performed to illustrate the differences in computing power and cost.

(ii) Cost and Frequency of Computations

The cost of the resource includes both the hardware and software cost, the physical infrastructure to house the resource and the miscellaneous cost to operate the resource, e.g.

[4] http://top500.org/

the electricity bill, cooling system, the personnel, insurance, etc. However, depending on the resource class, the provision of the exact same resource can have an effect in urgent computing cost. The cost of dedicated resource is mostly fixed but the cost of each urgent computation is related to the frequency of the computations and the number of cores/nodes used. In the case of public and on-demand resources, the cost is also related to the frequency of computations and the number of cores/nodes used.

In order to evaluate the cost of urgent computing on each resource class, the following symbols are defined. C_r represents the cost of a resource over a fixed period of time, e.g. a year. N_r represents the total number of cores-hours/nodes-hours used during the fixed time period. Thus, the computation cost of each core/node, C_n, is represented as follows:

$$C_n = \frac{C_r}{N_r} \tag{3.5.1}$$

Cost will be expressed in terms of an agreed upon cost unit, e.g. euros or US dollars. Cost per computation will be expressed in compute units. A *compute unit* refers to the minimum standard unit that a user will be charged for, e.g. one core per hour.

(class 1) Public Resources are an important class of resources that urgent computing can utilise. In order to leverage on such resources on demand, there is a high probability that existing running jobs have to be preempted to make way for the urgent jobs. The cost of urgent computation on such resources is represented as follows:

$$C_c = C_p + C_{qos} + C_u \tag{3.5.2}$$

C_p, C_{qos} and C_u refer to the cost of preemption, loss in quality of service (QoS) and urgent job respectively. The cost of preemption and loss of QoS are dynamic and are dependent on the running state of the targeted resource. C_p can be represented as shown in equation (3.5.3) and is strongly dependent on the jobs that will be preempted. Thus the decision on which jobs to preempt can strongly influence C_p. n_i and t_i refer to the number of cores/nodes used and the current wallclock time used by job i respectively. C_p is thus a sum of the cost of these preempted jobs. C_p can be simply seen as the direct cost of preemption.

$$C_p = (\sum n_i \cdot t_i) C_n \tag{3.5.3}$$

C_{qos} is the indirect cost of preemption and is a more difficult cost to measure and quantify. Generally, if more jobs and thus users are disrupted, the higher the perceived incurred indirect cost. Naturally other factors, e.g. time left until job is completed, possibility of resuming job instead of a complete restart, number of jobs per user being preempted, frequency of preemption, when the preempted jobs are restarted, will also influence the perceived loss of quality of service.

C_u can be similarly expressed as shown in equation (3.5.5). If all three classes of resources have the same resource type, C_n is typically the lowest for the public resource class. For a particular urgent event, C_u is dependent on the number of cores/nodes used

and the cost of each core/node. Since this resource class is shared, the cost per core/node is not greatly dependent on the urgent jobs since there are typically other jobs utilising the cores when urgent jobs are not active. However, there exist two additional costs C_p and C_{qos}, which are dependent on the C_n. Thus, the cost increases with increased frequency of urgent jobs. Consequently, public resources are more suitable for rarely occurring urgent events.

(class 2) Dedicated Resources are the most common class of resources that urgent use cases leverage upon. The cost of an urgent computation on such resources is represented as follows:

$$C_c = C_u \tag{3.5.4}$$

C_u refers to an urgent job. C_u is illustrated in equation (3.5.5) where n is the number of cores/nodes used, t is the wallclock time used (typically rounded up to next hour) and C_n is the computation cost of each core/node.

$$C_c = C_u = n \cdot t \cdot C_n \tag{3.5.5}$$

The cost C_n is dependent on N_r (refer to equation (3.5.1)). Since the resource class is dedicated, only urgent jobs will utilise the resource. Thus if more jobs and more cores are used, the cost per core/node C_n will be correspondingly be lowered. Dedicated resources are consequently most suitable for frequently occurring urgent events.

(class 3) On-demand Resources are pay-per-use resources where profit is expected since most resource providers are commercial entities. As such, in addition to the cost of resources, a profit variable, P_j, is included. P_j is modelled as a variable attribute that is dependent on the particular instance of computation since the cost of on-demand resources are dependent on the resource type, e.g. cores, memory and storage used, which is dependent on each computation. Similar to dedicated resources, there is no need to cancel/suspend computations. Thus, no C_p and C_{qos} are expected. C_u can also be similarly expressed as shown in equation (3.5.5). If all three classes of resources have the same resource type, C_n is typically the most expensive for the on-demand resource class .

$$\begin{aligned} C_c &= C_u \\ &= n \cdot t \cdot C_n \\ &= n \cdot t \cdot (P_j + C_{an}) \end{aligned} \tag{3.5.6}$$

where C_{an} refers to the actual cost per core/node without profit

Similar to public resources, C_c is directly proportional to the frequency of computations. The more you use, the more you pay. In principle, on-demand resources are also more useful for rarely occurring urgent events. If $n \cdot t \cdot P_j < C_c + C_{qos}$, on-demand resource is more cost effective than public resource and thus should be the selected resource class.

(iii) Intangible Criteria

Naturally, the choice of resource is often also swayed by the intangible criteria, e.g. personal preference, control, flexibility, etc. Assuming in the case where the cost of all three resource classes is the same and can fulfil the computation requirements, how should the resource classes be chosen? This is where the intangibles come in. Dedicated resources seem to be the preferred choice in such cases. This can be attributed to benefits such as ownership, direct control, flexibility to change configurations and the lack of usage restrictions, which is a common limitation of public resources. The need to serve multiple user groups led most public resources to impose user policies that restrict flexible usage of the resources. As such, to fulfil urgent computing on public resources, one has to adapt to the unique policies of each resource provider, leading to a significant overhead, in particular when more than one resource is involved. On-demand resource is thus arguably the second best class in terms of intangible criteria since it offers benefits similar to dedicated resource, with exception of ownership and direct control as in the case of dedicated resources. It shares also the benefits of public resources, where the maintenance and upgrade of the hardware are taken care of by a third party.

A Public Resource versus an On-demand Resource

In this dissertation, we will be targeting urgent uses where the frequencies of occurrences are low. Consequently, two classes of resources, public and on-demand, are appropriate candidates. Assuming that the computational requirements can be met by both classes due to similar hardware profiles, cost and frequency are the next deciding factors.

The costs of a public HPC resource, SuperMUC thin islands, from LRZ and two on-demand cluster options from Amazon Web Services (AWS)[5] that are hosted in the region EU (Frankfurt) in January 2015 with similar processor type, i.e. Sandy Bridge processors, are compared in Table 3.2. There are minor differences in the hardware profiles, number of cores or vCPU per node (third column), processor type (fourth column) in terms of versions, processor frequency (fifth column), memory (sixth column) and storage (seventh column).

The cost of a core per hour on the thin node island on SuperMUC is estimated (rounded up) to be 0.016 euros, i.e. 0.256 euros per node per hour. This estimation is calculated using the average annual funding for the system, its system software, direct system personnel, electricity, cooling system and mandatory independent commercial software, i.e. compilers, debuggers, etc. The total core-hours SuperMUC can offer per year is approximated from the usage statistic collected in 2014. This estimated cost does not include the building cost, non-system support personnel, extra commercial software, e.g. MATLAB, etc.

The information collected from AWS is known to be correct and up-to-date on 23 January 2015. Since the cost offered by AWS is based on US dollars, the cost is converted to euros using the exchange rate of 1 US dollars to 0.88 euros. This is the rounded live

[5] `http://aws.amazon.com/ec2/instance-types`

exchange rate provided by XE.com[6] on 23 January 2015.

Site	Type	Cores or vCPU	Processor Type	Processor Frequency (GHz)	Memory (GB)	Storage (GB)	Cost per Hour
LRZ	SuperMUC thin island	16	Intel Xeon E5-2680 8C	2.7	32	100 (NAS) + 1000 (GPFS)	0.256 €
AWS	m3.2xlarge	8	Intel Xeon E5-2670 v2	2.6	30	160 (SSD)	0.665 US$ (≈0.585 €)
AWS	c4.4xlarge	16	Intel Xeon E5-2680 v2	2.8	30	- (EBS)	1.032 US$ (≈0.909 €)

Table 3.2: Cost of SuperMUC and AWS

The per node-hour cost at LRZ is comparatively cheaper (refer to the last column of Table 3.2) than that offered by AWS since LRZ is a public resource provider and is thus not expecting a profit. However AWS resources have many intangible advantages and in particular the ability to be available on demand and thus swiftly without any additional and in particular manual interference from AWS. It also offers the flexibility and ease of setting up and configuring scalable resources as required by the user/use case without common policy restrictions that one expects from public resource providers. Naturally, the flexibility comes with some work. In the case of public resources, the maintenance of the resources, e.g. operating system updates, security patches, etc., are taken care of by the resource providers. In the case of AWS, they have to be managed by e.g. the operators.

Site	Type	Nominal Frequency	Peak Performance	Cores	Memory
LRZ	SuperMUC thin islands (Intel Xeon E5-2680 8C)	2.7 GHz	3.185 PFlop/s	147,156	288 TB
AWS	Amazon EC2 C3 Instance Cluster (Intel Xeon E5-2680v2)	2.8 GHz	0.59351 PFlop/s	26,496	0.105984 TB

Table 3.3: SuperMUC and AWS EC2 C3 Cluster

Table 3.3 further compares SuperMUC thin islands and the HPC machine AWS made in November 2014 for the Top500 list[7] in terms of nominal frequency, peak performance, number of cores and memory. In spite of the many advantages of on-demand resources, when HPC resources are required, public HPC resources like SuperMUC thin islands are simply computationally more powerful, i.e. significantly bigger number of cores, faster network, bigger and faster storage. As such, public HPC resources are a highly valuable class of resources for urgent computing.

Using the scenario presented in Genoa flash flood in 2014, Case Study 2 (refer to Section 6.2), where eight ensemble of forecasts requiring 640 cores each with a deadline of 3 hours. The computational cost C_c on the two resources, SuperMUC thin islands and Amazon EC2 C3 Instance Cluster, can be computed with equations (3.5.3) and (3.5.6) respectively.

[6] `http://www.xe.com/`

[7] `http://top500.org/system/178321`

Table 3.4 shows the resulting computation cost C_c (fifth column). The different cost components, C_u, C_p and C_{qos}, that contributed to C_c are illustrated in the second, third and fourth columns respectively. Since there is no QoS penalty at this moment for preempting jobs on SuperMUC, C_{qos} is not applicable. In the best cast scenario (first row), no jobs have to be preempted on SuperMUC ($C_p = 0$) if there are sufficient free slots for the eight ensemble forecasts to begin computing immediately. In the worst cast (second row), there are no free slots and the most expensive possible job, running its 48^{th} (maximum wallclock limit) computing hour and using the maximum number of nodes, 512, on the corresponding queue, has to be preempted. Naturally, the worst case can typically be avoided as shown in our work [13] since they are usually jobs that have just started and are using a smaller number of nodes to preempt. In the case of AWS EC2 C3 Cluster (last row), C_p and C_{qos} are not applicable. As such, the computation cost consists only of C_u, at 872.64 euros. The computation costs of both resources are insignificant when compared to the cost of the disaster, 303 million US dollars.

Resource	C_u	C_p	C_{qos}	C_c
SuperMUC thin islands (Best case)	245.76 €	0	-	245.76 €
SuperMUC thin islands (Worst case)	245.76 €	6291.46 €	-	6537.22 €
Amazon EC2 C3 Instance Cluster	872.64 €	-	-	872.64 €

Table 3.4: Computation Cost on SuperMUC and AWS EC2 C3 Cluster

In spite of the possible hurdles one has to overcome, the potential benefits of public HPC resources are too valuable not to investigate further. Consequently, this dissertation targets in particular this challenging class of resources. In order to facilitate the support of urgent computing on public resources, a fundamental change in policies is required, which can possibly only be achieved by political influences. A set of policy recommendations are thus included in Section 3.5.2 to establish a platform to initiate these policy changes.

3.5.2 Policy Recommendations for Public Resource Providers

The existing policies of most public resource providers are too restrictive to effectively support urgent computing. One of the crucial services, preemption, cannot be readily supported due to the policy restrictions. Concerns about security result in great resistance towards providing ubiquitous interfaces to expensive HPC resources. In fact, giving access to users outside the usage scope defined by the funding agency is in principle not allowed. In spite of the usefulness of frequency scaling (refer to Section 5.2.1) to urgent computing, due to high energy cost, evidence of application scalability is required before the option is activated. In order to address these restrictions, the following recommendations on three levels, funding agency, resource providers and users, are proposed. For urgent computing to be effectively supported, the funding agency needs to indicate the intent of supporting this class of computing. Next, the resource providers have to be involved to modify existing usage models and policies to enable urgent computing. Finally, the affected users

and urgent users must have up-to-date information on what system behaviour to expect and how to effectively use such resources.

Funding Agency

(i) Establish funding to enable public resource providers to allocate a part of the computing and human resources for urgent computing

(ii) Support policy changes that permit public resource providers to enable prioritised support to urgent computing

- Pool of urgent computing resources
- Facilitate worldwide accessibility when required
  - Establish guidelines for a generic public interface

Resource Providers

(i) Establish clear acceptable use policy for urgent computing

- Access requirements, regulations and policies
- Storage allocations and policies
- Computing resource limitations and policies
- Resource management system requirements and policies

(ii) Continuous monitoring and sharing of system status data, e.g. availability and reliability, of the computing resources and services

(iii) Enable usage of crucial services for urgent computing

- Preemption
- Frequency scaling
- Ubiquitous secured access

(iv) Regular testing of services and procedures utilised by urgent computing

(v) Establish clear security guidelines and rules to strengthen the protection without restricting usability

(vi) Establish a priority queue for urgent jobs where preemption is supported and jobs will start immediately

Users

(i) Clear documentation on use policy for normal users for a system or a part of system where urgent computing is enabled

- Possible job cancellation/suspension
- Accounting compensation and/or reduction rates
- Estimated delay
- Job restart/resumption

(ii) Clear documentation on use policy for urgent computing users to activate an urgent computation(s)

- Emergency contacts
- Technical documentation of the resource

(iii) Clear documentation from urgent computing users on their urgent system design

- Validation on scalability
- Possible risks, particular security, of the designed system
- Technical requirements on the resource
- List of services, tools and applications required on the resource

In this dissertation, the urgent system is designed while adhering to the recommendations. Best effort is made to ensure that the limitations, policies and requirements of the resources and resource providers are reasonably adhered to without compromising the core needs of the urgent system. This is to motivate public resource providers to readily support urgent computing by ensuring that there will not be major disruptions to their operation models.

Chapter 4

An Urgent System & A Task-based Ubiquitous Framework

The design of an effective urgent system is crucial to the success of urgent computing where completing the computations not only correctly but within the stipulated deadlines is required. In this chapter, an urgent system consisting of four components is shared in Section 4.1. To realise the urgent system, a task-based ubiquitous (TbU) framework as described in Section 4.2 is designed. This framework aims to address the chaotic and frequently unpredictable circumstances that entail disasters, and the need to manage multiple distributed heterogeneous computing resources to swiftly carry out the required computations, i.e. ensembles of forecasts.

Despite the advances in science and technology, it is still a challenge to accurately predict the onset and progress of many disasters. Thus, when a disaster occurred or is expected to occur in short order, the surrounding environments are expected to be dynamic and chaotic. It is thus reasonable to consider that a decision maker(s) who has to decide whether to initiate an urgent computation, analyse the urgent products and make crucial decisions to mitigate the loss, might not be in the vicinity of his/her work environment in these time-critical moments. Instead, easy to carry ubiquitous end user devices like mobile phones and tablets are more practical and have a higher chance of being readily available. The TbU framework thus aims to support such devices to enable urgent computing.

Additionally, the success of urgent computing depends on how swiftly the ensembles of forecasts can be prepared and initiated on multiple distributed heterogeneous computing resources. The TbU framework is thus designed with a three-layer architecture as shown in Section 4.2.1 to allow a separation of concerns, ubiquitous access, urgent computing and use case requirements, and heterogeneity of computing resources and their environments. The design of the urgent computer system based on the TbU framework is described in Section 4.2.2.

4.1 Urgent System

An urgent system is a system to enable urgent computing to support decision-making and coordination work such that loss mitigation is possible. This system can be represented as shown in Figure 4.1, consisting of four components, the operator, the urgent computer system, the decision & coordination system and controlled objects. The operator interacts directly with the urgent computer system and decision & coordination system, which have to share information with each other. The decision & coordination system has to also manage the controlled objects. The urgent system is designed with reference to the real-time system shown in Figure 2.1 [2]. The presence of multiple controlled objects and the non-deterministic characteristics of urgent computing imply a more complex decision and coordination process. Thus, a specific component for decision-making and coordination is designed.

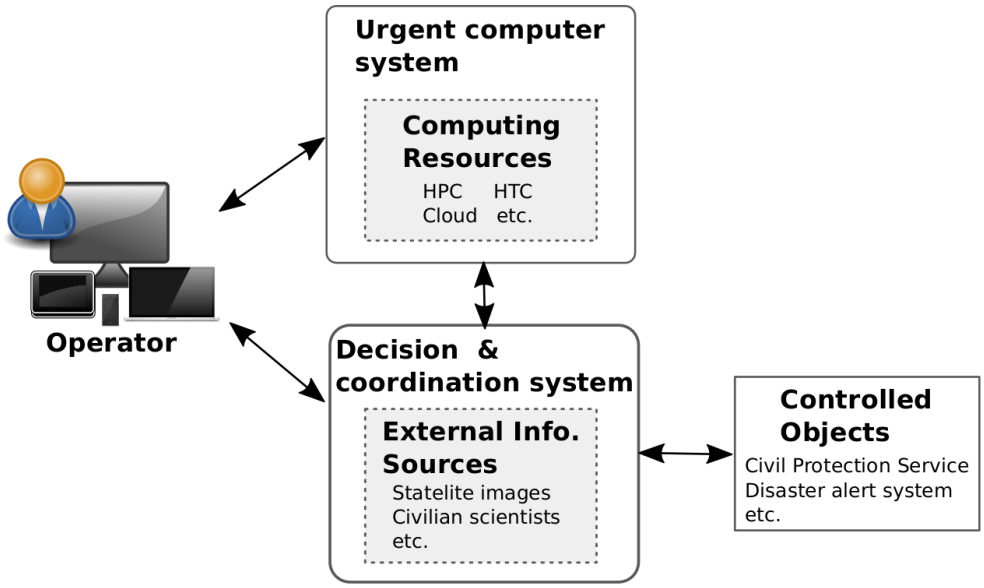


Figure 4.1: An Urgent System

The operator is typically an event domain scientist and/or civil protection officer who will decide if an urgent computation should be initiated. It is also possible for the operator to be an automated machine but due to the uncertainty in computing requirements until zero hour, it is more reasonable to anticipate a human operator.

The urgent computer system consists of computing resources to carry out urgent computations, i.e. forecasts. The requirements of urgent computing and computational requirements of each forecast are handled within this component. The underlying resources can be a single workstation, on-demand, HTC and/or HPC depending on the expected workload. Security and administrative requirements between the operators and the resources, data collection for computations, allocating resources to urgent computations, etc. are some of the important functions under the urgent computer system's jurisdiction. SPRUCE,

UrbanFlood Common Information Space (CIS) [30] and TbU framework implementation illustrated in Section 4.2.2 are examples of an urgent computer system.

The decision & coordination system is another critical component of the system. It assists the operator in making informed mitigation decisions supporting the evaluation of urgent products from the urgent computer system. Once a decision is made, it facilitates the operator to effectively disseminate the decision to the controlled objects such that mitigation activities are carried out. Since the circumstances entailing an urgent event are often very dynamic and chaotic, decisions need to be updated as new information comes in. Thus, this component must be able to coordinate live updates to all controlled objects.

The controlled objects can be mechanical (sensors) or humans. Examples of controlled objects are civil protection services, i.e. firefighters, police and doctors, evacuating civilians, disaster alert systems, news agency, etc. Information exchange between the controlled objects and the decision & coordination system is crucial to ensure that errant controlled objects and unforeseeable developments are appropriately managed.

4.2 Task-based Ubiquitous Framework

A generic TbU framework is correspondingly designed to realise the urgent computer system, the focus of this dissertation. This framework will enable ubiquitous access to the underlying heterogeneous distributed computing resources, easy management of their dynamic environments and the swift adoption of new resources and use cases while ensuring the stipulated deadlines are met. It allows urgent computing and use case specific computing requirements to be handled separately such that the use case specific activities, which are expected to require zero hour modifications as requirements may vary from event to event, can be independently updated.

4.2.1 Architecture

The architecture of the urgent computer system based on the TbU framework is illustrated in Figure 4.2. It is based on top of a three-layer architecture consisting of a ubiquitous process orchestrator (Layer 1), an activity & task manager (Layer 2) and a subtask coordinator (Layer 3) illustrated with pink, green and yellow rectangles respectively. Within the activity & task manager, four urgent software component managers (the blue rectangles), information, resource & environment, schedule and fault, are designed to enable the framework to manage the requirements, in particular the time constraints, of urgent computing.

The three layers are specifically designed to enable a separation of concerns when enabling the following functionalities.

(i) Layer 1 – To provide an HMI interface for operators to swiftly and easily access the computing resources.

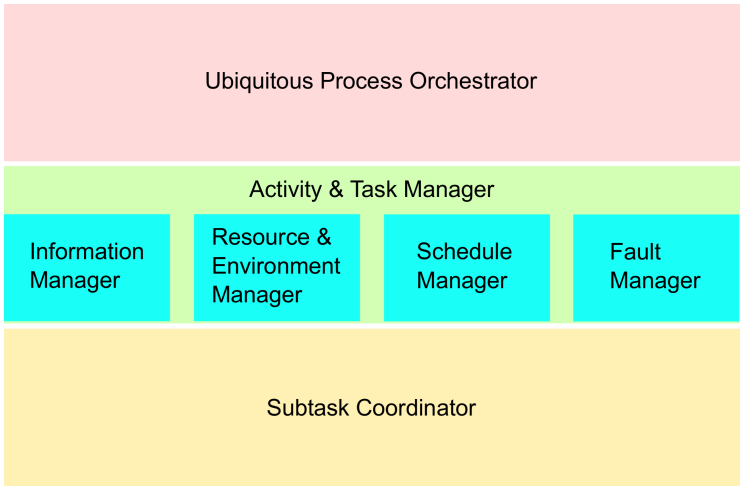


Figure 4.2: Three-Layer Architecture

(ii) Layer 2 – To provide an easy to configure and adaptable platform for urgent computing requirements and use case specific workflows to be represented.

(iii) Layer 3 – To provide a simple integration interface to manage the heterogeneity of computing resources and their environments.

Layer 1 manages the client requirements while Layer 2 handles the use case and urgent computing specific requirements. Layer 3 takes care of the heterogeneity of the distributed computing systems and their environments.

To assist each layer in functioning effectively, the work steps within this architecture are further represented by processes, activities, tasks and subtasks based on the concepts of work breakdown structure [105] in systems engineering and the IEEE standard for developing software life cycle (SLC) processes [106, p. 11-12]. The incremental breakdown of a system into smaller components enables better organisation and management of each identified functionality. It also complements the layered architecture such that the lower layers get an incrementally refined division of work to address the requirements from urgent computing, use cases and heterogeneous computing resources.

The IEEE standard for SLC processes is also leveraged on to define the work steps breakdown. In this standard, a *process* is defined to be composed of activities, an *activity* is a constituent task of a process and a *task* is the smallest unit of work subject to management accountability. Since 1997, the standard [107] is updated to use the term "activity group" as opposed to "process".

The breakdown of work steps in this dissertation are defined as such.

- A **process** consists of a sequence of general activities to fulfil the main functionality of a system.
- An **activity** is a major piece of general work that contributes to the goal/functionality of a process.

- A **task** is a minor piece of work, i.e. coarse grain task, which describes an activity specific to its application, e.g. to the use case and/or urgent computing requirements. An activity can consist of many tasks. In addition, a task can be further extended into one or many subtasks, i.e. a fine grain instance of a task.
- A **subtask** represents a technical implementation of a task for a specific work assignment.

The processes represent the main functionalities, e.g. start urgent computations and analyse urgent products, which the operators are required to manage within Layer 1 via ubiquitous end user devices. Activities are used to describe the processes in more details in Layer 2 without considering the underlying resources. Tasks are then introduced to describe the activities based on urgent computing and use case specific requirements. General descriptions with reference to the computing resources are introduced to ease the integration with Layer 3. Sub-tasks in Layer 3 represent the resource specific implementations of the tasks where the heterogeneity of distributed computing resources and their environments are managed.

The relationship among processes, activities, tasks and subtasks is elaborated in Figure 4.3 using the same colour scheme as the layer they belong to in the three-layer architecture. 1 to n processes (refer to the pink rounded rectangles) can be defined for a use case. These processes correspond to Layer 1 of the TbU architecture and will thus be orchestrated by the ubiquitous process orchestrator. Each process can be refined into 1 to n activities (refer to the green rectangles), which can take place sequentially or in parallel. Each activity can be further broken down to 1 to n tasks (refer to the blue elongated hexagons), which can run sequentially or in parallel. The activities and tasks are a part of Layer 2 of the TbU architecture and are managed by the activity & task manager. Finally, each task can be further decomposed into 1 to n subtasks (refer to the yellow ovals). The subtasks are coordinated by Layer 3 of the TbU architecture, the subtask coordinator.

The three-layer architecture allows a separation of concerns to enable extensibility and maintainability. Details of each layer will be elaborated in the following subsections.

Layer 1 – Ubiquitous Process Orchestrator

Layer 1, the ubiquitous process orchestrator, administers the human-machine interface (HMI). The crucial ubiquitous feature facilitates interaction with a variety of heterogeneous computing resources via ubiquitous end user devices, which have the highest probability of being readily available in the event of a disaster. This enables the urgent system to be accessible from anywhere at anytime.

The ubiquitous process orchestrator does not manage any client software and will only provide generic ubiquitous process interfaces to orchestrate the processes and activities of urgent use cases. There is an implicit requirement that the selected client software must function on selected ubiquitous end user devices but that is outside the jurisdiction of the ubiquitous orchestrator. The clear separation of concerns between the layers ensures that there is no direct coupling between the ubiquitous orchestrator and the subtask coordinator.

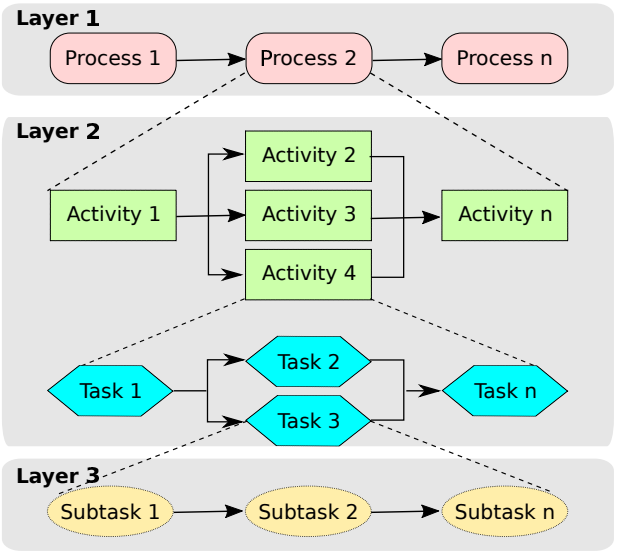


Figure 4.3: Relationship among Processes, Activities, Tasks and Subtasks

Thus, any changes on the underlying resources, will have minimal impact on this layer and the clients.

Layer 2 – Activity & Task Manager

Layer 2, the activity & task manager, provides the interface to marshal any combination of tasks to realise the activities of urgent use cases. Each activity will be described as a sequence of tasks in this layer. Existing work steps of event domain scientists will be defined as tasks to be managed by this manager.

The most crucial components within this layer are the four urgent software component managers, information manager, resource & environment manager, schedule manager and fault manager. These managers are responsible for ensuring that the urgent computing requirements are fulfilled within the urgent system. They consist of mandatory tasks that have to be utilised by all use cases. Their main responsibility is to ensure that the defined activities are completed by adhering to the urgent computing requirements before the stipulated deadline. The duties of each manager are elaborated as follows:

(i) Information Manager

The function of the information manager is to administer the task to collect data. The process model of this task is shown in Figure 4.4. Information about the computing resources and requirements of the urgent use cases has to be identified, collected, processed and stored via time-triggered monitoring in Phase 1. The requirements of the use case, e.g. the required number of computing nodes per resource, are used to make an initial selection on the set of appropriate resources in Phase 2. The resource set is further refined with the

constraints given by the operator, e.g. choose resources where the application required is pre-installed, in Phase 3. The criterion function fulfils mandatory requirements while the utility function manages preferable constraints.

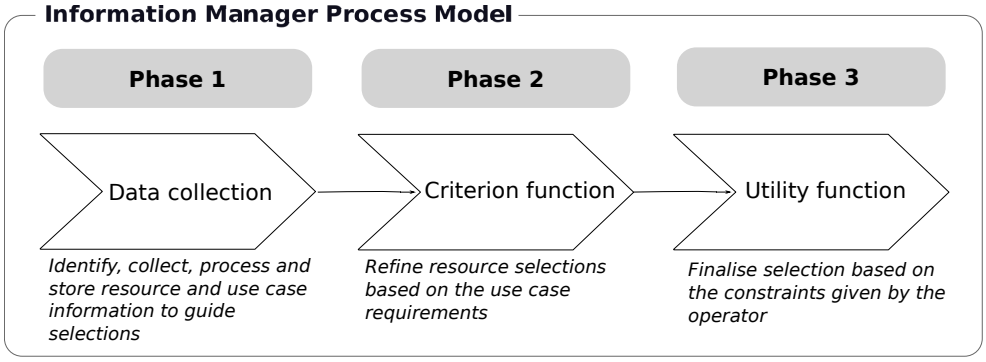


Figure 4.4: Process Model of Information Manager

Phase 1 of this manager typically takes place in advance. Data collected is processed and stored in information systems for easy retrieval. Phase 2 and 3 are initiated by an operator when an urgent event is expected.

(ii) Resource & Environment Manager

The resource & environment manager administers the prevalent statuses, e.g. availability, of resources and their dynamic environments. The process model of this manager is shown in Figure 4.5. Pre-collected data on the selected resource set is retrieved from the information manager in Phase 1. Each resource in the resource set is additionally verified by the resource & environment manager on the fly, i.e. event-triggered monitoring, to ensure that the existing statuses are still adhering to the requirements and constraints provided by the operator. This is crucial as the environments on resources are typically dynamic. Resources that no longer satisfy the requirements, in particular the criterion function, e.g. partial maintenance of the resource and consequently insufficient nodes for the urgent computation, will be removed from the resource set in Phase 2. In phase 3, the constraints set by operators are used to finalise the selection. The three phases in this manager are triggered by the information manager.

(iii) Schedule Manager

The schedule manager is responsible for scheduling the urgent computations, e.g. preemptive scheduling [13], on the given resource set provided by the resource & environment manager. The process model of this manager is shown in Figure 4.6. This manager will evaluate the information collected by the information manager and the resource set given by the resource & environment manager in Phase 1. The most crucial urgent computing

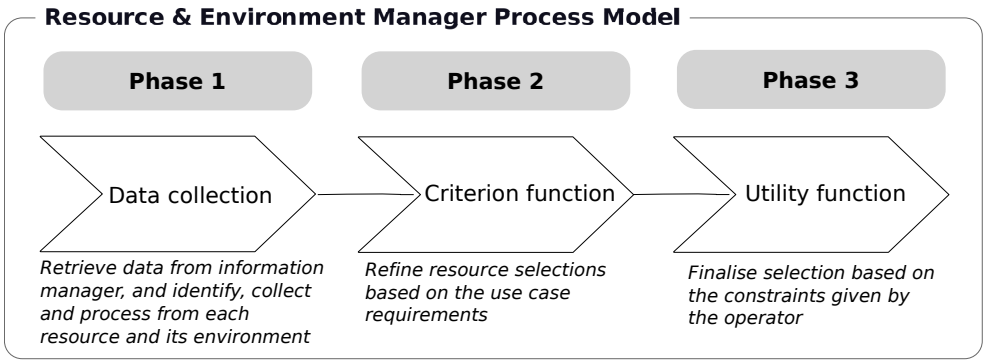


Figure 4.5: Process Model of Resource & Environment Manager

requirement, deadline, is used by the criterion function to select the the most appropriate resources from the set in Phase 2. The utility function is used additionally to ensure that the non-mandatory constraints, e.g. reliability and energy usage, are also fulfilled in Phase 3. The final selected resources will be assigned the urgent computations.

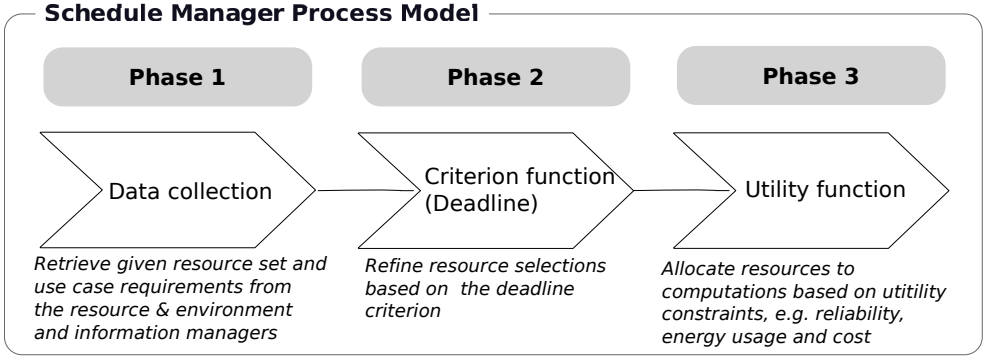


Figure 4.6: Process Model of Schedule Manager

The three phases of this manager are triggered by the resource & environment manager. This manager will in turn trigger the fault manager.

(iv) Fault Manager

The fault manager improves the reliability of the urgent system by providing mechanisms to recover from pre-defined faults. The faults are classified into two categories, benign and malignant, based on on their impact on meeting the stipulated deadline. These two categories are then refined to include the characteristics, recoverable and irrecoverable, to reflect the ease of recovery. They are defined as shown in the following list.

- **Benign faults** are harmless faults that will not have an adverse impact on meeting the stipulated deadline.

  - **Benign *recoverable* faults** are harmless faults that can be easily recovered and will not have adverse impact on meeting the stipulated deadline.
  - **Benign *irrecoverable* faults** are harmless faults that cannot be recovered and have to be rescheduled by the schedule manager but will not have adverse impact on meeting the stipulated deadline.

- **Malignant faults** are harmful faults that will result in missing the stipulated deadline.
  - **Malignant *recoverable* faults** are harmful faults that upon recovery will result in missing the stipulated deadline.
  - **Malignant *irrecoverable* faults** are harmful faults that cannot be recovered and have to be rescheduled by the schedule manager, which result in missing the stipulated deadline.

Undefined faults will be classified as benign irrecoverable faults and will be rescheduled. The process model of this manager is shown in 4.7. When a fault is detected, the information manager and the resource & environment manager are contacted to collect more information about the fault in Phase 1. The list of pre-defined faults will be retrieved from the information manager while the resource & environment manager will check the affected computing resource and its environment for more data about the fault. The gathered data will be used to analyse the fault based on the deadline, benign or malignant, criterion in Phase 2. Finally the fault is classified in Phase 3 and appropriate recovery measures are carried out. If a reallocation of the affected application(s) is necessary, e.g. irrecoverable benign faults, the schedule manager will be contacted. All three phases of this manager are triggered when a fault is detected.

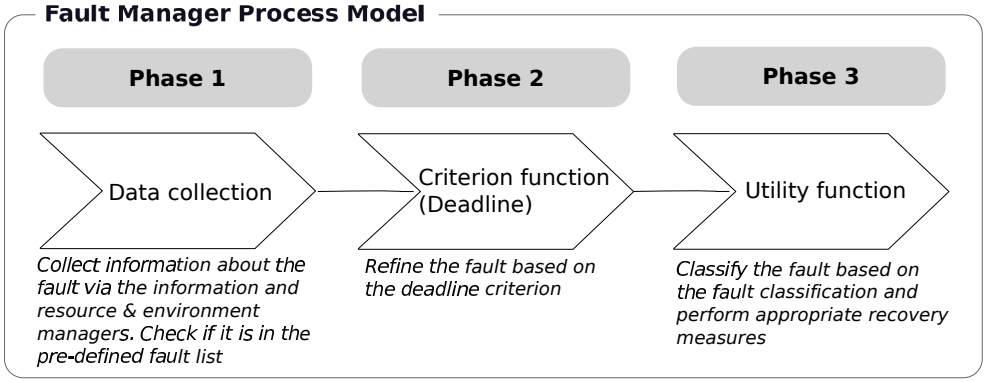


Figure 4.7: Process Model of Fault Manager

Layer 3 – Subtask Coordinator

Layer 3, the subtask coordinator, coordinates the complex interactions with the heterogeneous computing resources and their environments. Four fundamental subtask functions,

AUTH, EXEC, TRAF and TERM, are defined to describe and execute each task. These subtask functions are identified by reviewing the workflow of different use cases, both urgent and non-urgent, while collaborating with different domain scientists in projects, e.g. EGI, DRIHM and VERCE. Most if not all urgent computing activities can arguably be described with this set of subtask functions.

(i) **AUTH** – Authenticate

(ii) **EXEC** – Execute

(iii) **TRAF** – Transfer

(iv) **TERM** – Terminate

The AUTH function manages the connection and authentication to the underlying computing resources. The EXEC function executes the tasks required by the use cases. The TRAF function coordinates data transfers and exchanges. The TERM function manages the termination of ongoing AUTH, EXEC and TRAF activities. The tasks, work steps of scientists, will be expressed and implemented via these subtask functions. These functions will enable a rapid integration of existing work steps of an urgent use case into the TbU framework. Heterogeneity in computing resources and their environments will be handled within these functions.

4.2.2 Design of the Urgent Computer System

To enable the urgent computer system to support ensembles of forecasts, the TbU framework is used to design the prototype. The resulting TbU compute process is shown in Figure 4.8. The urgent software component managers, Information, Resources & Environment, Schedule and Fault managers, will be abbreviated as Info Manager, R&E Manager, Sched Manager and Fault Manager respectively in the figures.

The urgent workflow can be generalised into three processes, compute, analyse and decide (refer to the pink rounded rectangles in Figure 4.8). The compute process describes the allocation of ensembles of forecasts on a given set of computing resources while the analyse process shows the evaluation activities of the computed results. The decide process, the final step of the workflow, represents the mitigation decision-making activities. The compute process will be the main focus of this dissertation. The analyse process is realised with a ubiquitous visualisation service[1] (refer to Appendix A), which has since been adopted as a production service at LRZ. The decide process is assumed to be a manual process that is performed by the event domain scientists and/or civil protection officers by evaluating the urgent products.

The compute process is broken further into three activities, collect, allocate and retrieve as elaborated in Figure 4.8 with green rectangles. The collect activity represents

[1] https://rvs.lrz.de

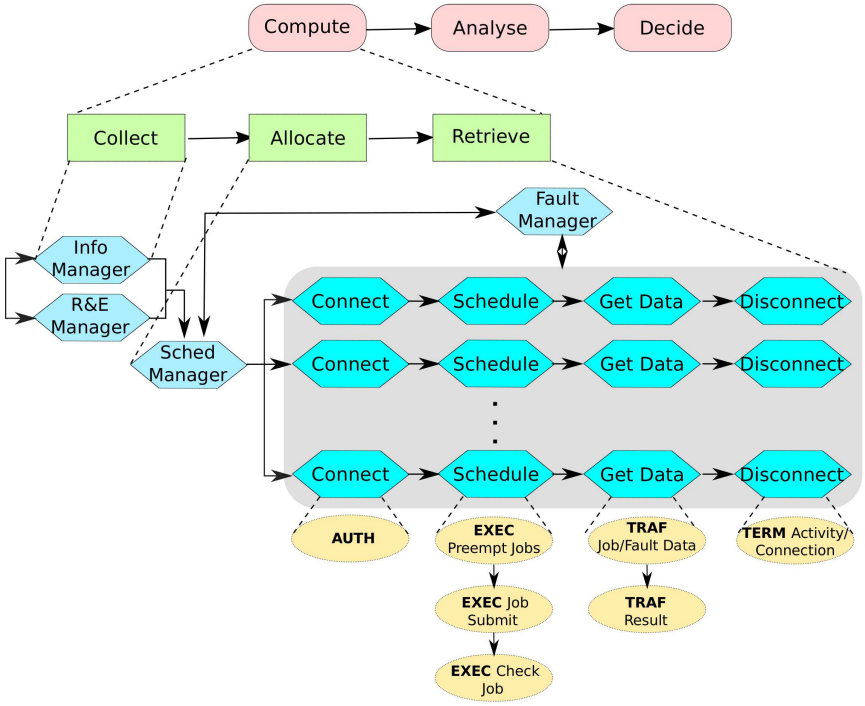


Figure 4.8: TbU Compute Process for Ensembles of Forecasts

the collection of data, e.g. use case and resource requirements from the operator, and information of the computing resources and their environments from the information and resource & environment managers (blue elongated hexagons on the left). The allocate activity describes the work performed by the schedule manager to allocate resources for the ensemble of forecasts. The fault manager will monitor the computing resources for faults and rectify them when possible. Finally, upon the completion of the computations, the results are retrieved for the analyse process.

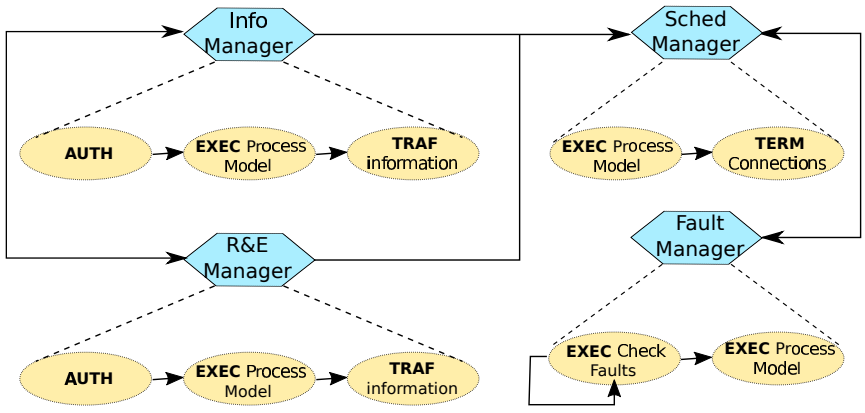


Figure 4.9: TbU Task Diagrams of the Urgent Software Component Managers

The core tasks performed on the computing resources are enclosed within the gray box

in Figure 4.8. Multiple concurrent instances of the task thread, connect, schedule, get data and disconnect, are initiated by the schedule manager (refer to the blue elongated hexagons in the gray box). Each instance represents a forecast in the ensemble. For each forecast, a connection is initiated on the allocated computing resource and the forecast job is scheduled on it. In the case of using shared resources, if there are insufficient available cores/nodes, running jobs will be preempted to free cores/nodes, i.e. preemptive scheduling, for the allocated forecast to commence immediately. Two possible approaches to realise preemptive scheduling are shared in Appendix B [13]. Data is retrieved to check the status of the job and for possible faults. Once the job is completed, the result is retrieved and the connection is terminated. Each of the core tasks is further described by the fundamental subtasks. This allows the heterogeneity of computing resources to be managed within the subtasks.

The TbU task diagram illustrating the interactions among the four urgent component managers (blue elongated hexagons) is shown in Figure 4.9. Each manager is presented as a task that is refined into a number of subtasks (yellow ovals). To simplify the illustration, the defined process models of the managers are depicted as subtasks. Both the information manager and resource & environment manager are expected to provide information of the resources and their environments, and resource allocation requirements to the schedule manager. The resource & environment manager is additionally expected to manage the dynamism of resources and their environments, and to provide the schedule manager with the most current information and requirements. The fault manager also receives updates of the allocated resources and forecasts from the resource & environment manager. If faults that require reallocation are detected, the fault manager will provide the schedule manager with updated information on the resources and their environments, and resource allocation requirements. The schedule manager thus manages the dynamism in resources and environments based on the inputs it received. It will allocate the required forecasts to computing resources swiftly while ensuring that the timing constraints and other requirements it received are taken into consideration. A set of resource allocation heuristics, which is discussed in detail in Chapter 5, is designed to enable the schedule manager to manage ensembles of forecasts.

In summary, the TbU approach enables the urgent computing framework to coordinate multiple forecasts simultaneously under time constraints on multiple distributed heterogeneous computing resources. It supports ubiquitous access to facilitate the interactions with multiple forecasts and resources from anywhere anytime. Ubiquitous access is considered as crucial in view of the chaotic environments that typically accompany an urgent event. The task-based feature provides the adaptability to refine the forecast models when additional insights are gained from zero hour data.

Chapter 5

Resource Allocation Heuristics

Enabling multiple forecasts with varying execution times to complete within a deadline is a challenging task. As such, a heuristic approach is employed to manage the resource allocation. In Section 5.1, the obligations of urgent computing for ensembles of forecasts and the objectives of the resource allocation heuristics, maximising makespan robustness, minimising energy usage and maximising reliability across resources and/or sites, are identified.

The allocation must meet the deadline in face of perturbations and reliability issues of using multiple computing resources while adhering to the realistic resource requirements, i.e. efficient energy usage when frequency scaling options are used. Two mathematical models to quantify robustness in terms of makespan, and reliability in terms of resource and site are also developed and shared in Section 5.2. The influence of frequency scaling and thus energy usage is modelled within the robustness model.

The makespan of an allocation is however dependent on the forecast allocation patterns, concurrently, consecutively or a combination of both, which are utilised. The allowed allocation pattern is in turn dependent on the number and size of computing resources available. Section 5.3 thus illustrates the three possible allocation patterns, the associated resource numbers and sizes, and how they influence makespan.

A set of resource allocation heuristics is shared in Section 5.4 to incrementally fulfil the identified objectives based on priority. The effectiveness of an allocation is quantified using the robustness and reliability models and the calculated ETS values for a distributed set of computing resources. In order to allow further assessment and comparison of the derived results and metrics, an assessment model is introduced in Section 5.5. The final aim is to provide a robust and reliable resource allocation heuristic for ensembles of forecasts that enables efficient energy usage under time constraints.

5.1 Obligations and Objectives of Resource Allocation Heuristics

The decision to select an optimal resource allocation for ensembles of forecasts is either P-complete or NP-complete as it involves a number of objectives, which cannot be simultaneously optimal. Thus a heuristic algorithmic approach is applied to deal with this allocation problem. In principle, the resource allocation must ensure that the following practical obligations are met.

(obligation 1) Meeting the stipulated deadline

(obligation 2) Maximising the number of successfully allocated and completed forecasts

Meeting the stipulated deadline **(obligation 1)** is the most fundamental criterion of urgent computing. However, the earlier the forecasts complete their computations, the more lead time there will be for decision-making and mitigation activities. As such, minimising the makespan (refer to Section 1.5 for definition) of a resource allocation is a natural objective. Since the makespan is estimated by considering the execution time of each forecast on the allocated resource, errors have to be expected. The resources in question can have perturbations that affect the actual execution times. To minimise the effect of such errors, makespan robustness, which takes into consideration the errors introduced due to perturbations is used instead as one of the objectives for the heuristics to optimise. *Makespan robustness* refers to the tolerance of an urgent system to perform faithfully in event of perturbations such that the length of the schedule of a resource allocation adheres to the deadline.

Frequency scaling, a technique to ramp up the processor frequency to improve the computing performance (faster) at the expense of electrical power (energy), is an option available on many modern computers. Levering on this option will have a positive influence on the makespan of an allocation and is thus a valuable feature to be utilised. However, since it comes at the expense of energy (increasing energy cost), most public resource providers are either reluctant to offer this functionality or are offering this functionality with prerequisites due to their power constraints. At LRZ, the efficient use of the particular resource and thus energy has to be proven before this option is activated for a particular user. Consequently, the selection of an optimal frequency while minimising energy usage is another objective of the heuristics.

Maximising the number of successfully allocated and completed forecasts **(obligation 2)** before a stipulated deadline is a basic criterion while coordinating ensembles of forecasts with timing constraints. The realisation of this obligation is dependent on the available resource set, the number of resources available and the sizes of the resources (refer to Section 5.3). There is an obvious and yet practical method to improve the success rate by selecting more reliable resources for the forecasts. However, this can lead to an oversubscription of a few resources, which can result in a single point of failure. Thus, reliability

of a resource allocation has to take into consideration the distribution of forecasts across resources and/or computing sites as the final objective to optimise via the heuristic approach.

In summary, the objectives to fulfil are as follows:

(obj. 1) Maximise makespan robustness

(obj. 2) Minimise energy usage

(obj. 3) Maximise reliability across resources and/or computing sites

In order to quantify makespan robustness, a robustness model specific to urgent computing is developed. Within this model, the frequency scaling option is included since it influences the makespan. Similarly, a reliability model is also implemented to quantify the reliability of an allocation across resources and sites based on our requirements. Both models are elaborated in the next section.

5.2 Robustness and Reliability Models

Both models are expressed mathematically based on the research work in [108][65] and applied specifically to urgent computing. The robustness model aims to quantify the robustness of an allocation in terms of makespan. The characteristic of the targeted computing resources, frequency scaling (energy usage), is included in the the makespan robustness model as they influence the makespan of the selected allocation. The reliability model attempts to measure the reliability of an allocation to enable a distribution of forecasts across resources and sites.

5.2.1 Robustness Model

Robustness is defined as the tolerance of a computing system to perform faithfully in event of perturbation(s). Perturbations, e.g. network issues resulting in reduced input-output read/write speed, can lead to performance degradation that affects the execution times of computations and consequently the makespan of allocations. This can cause dire consequences as a result of missing deadlines. *Makespan* refers to the length of a schedule, i.e. from the start of an allocation, t_s, to the point in time when all allocated urgent computations complete, for a given resource allocation while *execution time* refers to the amount of time a single forecast takes to complete its computation on a given resource.

To ensure that the impact of perturbations is minimised, designing robustness into the system is of utmost importance. Based on [108], a mathematical metric to measure robustness can be designed via a four-steps FePIA (named after the steps as created by the authors of [108]) procedure as shown in the list below. The FePIA procedure is thus applied to enable urgent computing to support ensembles of forecasts.

(i) Identify the system performance features Φ based on the selected robustness criterion.

(ii) Identify possible system and environment perturbation parameters.

(iii) Identify the impact of perturbation parameters.

(iv) Determine the smallest collective variation in perturbation parameter values.

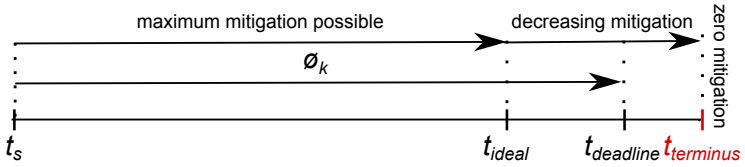


Figure 5.1: Urgent Computing Timing Variables

The most crucial robustness criterion in urgent computing is deadline. The timing variables, t_{ideal}, $t_{deadline}$ and $t_{terminus}$ (refer to Section 3.3), of urgent computing deadlines will be utilised in the model. Makespan ϕ_k is identified as the system performance feature to limit such that the deadline criterion can be met robustly. Maximising makespan robustness will implicitly imply minimising the makespan of an allocation. The selected resource allocation μ must satisfy the initial condition as shown in equation (5.2.1). Figure 5.1, which is based on Figure 3.4, summarises the relationship among the defined timing variables and the variables in the model, t_s and ϕ_k.

$$t_s + \phi_k < t_{deadline} \qquad \text{and } t_{deadline} < t_{terminus} \tag{5.2.1}$$

where

ϕ_k represents the kth performance feature in Φ.

In principle, the equation can accommodate more than one performance feature. However, if more performance features are included, i.e. $k > 1$, the computation complexity will increase exponentially. Consequently, only one performance feature is included in each model and near-optimal heuristics are leveraged on instead.

System and environment conditions of each computing resource c_j contribute to the set of identified perturbation parameters π_j that can adversely influence makespan as shown in equation (5.2.2).

$$\phi_{kj} = f(\pi_j) \qquad \text{and } \pi_j \in \Pi \tag{5.2.2}$$

where

ϕ_{kj} refers to the makespan of a given computing resource c_j for resource allocation μ.

$f(\pi_j)$ represents the maximum execution time of the allocated forecast applications a_i on a given computing resource c_j

Some examples of pertubations are "bad" computing nodes with lower computing performances than average and faulty batch schedulers, which under-allocate the required number of computing nodes leading to simultaneous multithreading (SMT). Both can result in lower computing performance, i.e. longer execution time, without affecting the ability to complete the computation.

The maximum execution time $f(\pi_j)$ of a computing resource is dependent on how the computations are allocated, consecutively and/or concurrently. Section 5.3 illustrates the possible allocation patterns and how it affects $f(\pi_j)$.

The makespan of a resource allocation μ on n resources is as follows:

$$\phi_k = \max_{\forall c_j} f(\pi_j) \qquad \text{and } j = 1..n \tag{5.2.3}$$

In order to satisfy the identified system performance feature, makespan, the boundary condition that illustrates the error tolerance has to be defined. The acceptable tolerance has to satisfy equation (5.2.1). The ideal boundary condition is $\phi_k \leq t_{ideal}$. However realistically, as long as the makespan is within the limits as shown in equation (5.2.4), mitigation activities are possible.

$$t_s < \phi_k < t_{terminus} \tag{5.2.4}$$

The error tolerance has to be set within the above range, using the minimum case as follows:

$$\phi_k^{actual} = \tau \phi_k^{estimated} \qquad \text{and } \phi_k^{actual} \leq t_{deadline} < t_{terminus} \tag{5.2.5}$$

where

ϕ_k^{actual} refers to the actual makespan of a resource allocation μ.

$\phi_k^{estimated}$ is makespan of a resource allocation μ by using pre-collected/computed data of each forecast a_i on each computing resource c_j.

τ is the tolerance value, i.e. tolerable error, for a given resource allocation. τ is ≥ 1.

Finally, the Euclidean norm (l_2-norm) is applied to find the largest distance between the actual and estimated perturbation parameters, π_j^{actual} and $\pi_j^{estimated}$ respectively. This distance represents the tolerable error range while adhering to the deadline. Minimising it will imply minimising makespan and maximising the tolerance towards perturbation errors of a computation on a given resource. This distance will be referred as the *robustness radius* r_μ as shown in equation (5.2.6).

$$r_\mu(\phi_k, \pi_j) = \min_{\pi_j : \phi_k = \tau \phi_k^{estimated}} \| \pi_j^{estimated} - \pi_j^{actual} \|_2 \tag{5.2.6}$$

The above equation can be further simplified since the actual makespan π_j^{actual} can be taken as a point while the estimated makespan $\pi_j^{estimated}$ as a hyperplane. Consequently, the

right-hand-side of equation (5.2.6) can be solved by a point-to-plane distance formula [109, p.22][108, p.633] as shown in equation equation (5.2.7).

$$\begin{aligned} r_\mu(\phi_k, \pi_j) &= \frac{\tau\phi_k - f(\pi_j)}{\sqrt{\text{number of consecutive applications allocated to } c_j}} \\ &= \frac{t_{deadline} - f(\pi_j)}{\sqrt{\text{number of consecutive applications allocated to } c_j}} \end{aligned} \tag{5.2.7}$$

Finally, to extend the above robustness radius equation for all $\phi_k \in \Phi$, the robustness metric ρ_μ, which is the minimum of all robustness radii (the worst case robustness radius for a given allocation) is defined as follows:

$$\rho_\mu(\Phi, \pi_j) = \min_{\phi_k \in \Phi} r_\mu(\phi_k, \pi_j) \tag{5.2.8}$$

The goal is to maximise makespan robustness radius for a given resource allocation μ. Maximising the robustness radius is basically minimising makespan such that the stipulated deadline is met **(obligation 1)** even in event of system perturbations. Additionally, it also maximises the lead time for decision-making and mitigation activities.

Frequency Scaling and Energy Usage

Frequency scaling is a technique to ramp up the processor frequency to improve the computing performance (faster) at the expense of electrical power (energy). It is an option available on many modern computers. However, it is typically not enabled by default on public resources due to the higher energy usage and thus higher energy cost. Only applications, which have proven that they can efficiently take advantage of higher frequencies will be given the privilege.

In the case of urgent computing, higher frequencies can potentially minimise makespan and result in more lead time for decision-making and mitigation activities. It is thus a critical feature that cannot be ignored for computations with time constraints. Equation (5.2.7) is modified to include the frequency scaling as follows:

$$r_\mu(\phi_k^{makespan}(freq), \pi_j) = \frac{t_{deadline} - f(\pi_j)}{\sqrt{\text{number of consecutive applications allocated to } c_j}} \tag{5.2.9}$$

This however will be an incomplete representation if energy usage is not represented. In order to minimise makespan (maximise makespan robustness), the highest frequency of each resource will inevitably be chosen. This will lead to energy wastage since not all computations need to run at the highest frequency to minimise the makespan (refer to equation (5.2.8)).

For example, in the case where there are two computations to be performed, computation A is estimated to require 2 hours while computation B needs only 1 hour using the maximum possible frequencies on the respective resources. If computation B uses the

default (the lowest) frequency, it will complete its computation in 1.5 hours. In this case, it is unnecessary for computation B to utilise the highest frequency since it will have to wait for computation A to complete before the decision-making and mitigation activities can be carried out. Thus, better energy utilisation can be achieved by allocating computation B with the default frequency.

To reflect this frequency scaling-energy relationship when allocating resources for ensembles of forecasts, energy usage is included as an objective in the heuristics defined in Section 5.4. Energy usage is measured in terms of energy-to-solution (ETS) [110]. *ETS* is the total energy in kilowatt hours (kWh) required for a computation to arrive at its solution.

5.2.2 Reliability Model

Reliability is the ability of a computing system to perform the required functions correctly within a specified time period. Unlike robustness, reliability does not manage perturbations in the computing system but is mainly concerned with the trustworthiness of a system during normal, i.e. expected, operation. A reliable resource allocation should avoid a single point of failure by selecting a good distribution of forecasts across reliable resources across sites.

Reliability in our reliability model will be a function of availability. If a system is frequently unavailable over a planned operation period, i.e. a time period where no maintenances or downtimes are expected, reliability will correspondingly decrease. The reliability/availability of services, such as availability of the batch schedulers, which are required to fulfil the urgent computations will not be considered in this dissertation but can be included by an appropriate redefinition of $\phi_j^{reliability}$. Reliability of a resource can be represented as shown in equation (5.2.10).

$$\phi_j^{reliability} = \frac{A_j(t)}{P_j(t)} \tag{5.2.10}$$

where

- $\phi_j^{reliability}(t)$ represents the performance feature, reliability, of a resource c_j over a time period t.
- $P_j(t)$ refers to the planned operation of resource c_j over time period t.
- $A_j(t)$ denotes to the actual operation of resource c_j over time period t.

The resource-based reliability radius and metric of a given allocation μ are defined by using the same procedures as the robustness makespan [18] without a robustness criterion. The corresponding reliability radius derived with Euclidean norm (l_2 norm) is shown in equation (5.2.11).

$$r_\mu^{resource}(\phi^{reliability}, t) = \frac{\phi_j^{reliability}}{\sqrt{\text{number of applications allocated to resource } c_j}} \tag{5.2.11}$$

where

$r_{\mu}^{resource}(\phi^{reliability}, t)$ refers to the reliability radius of a given resource allocation, μ, of resource c_j at time t.

Since one computing site can host more than one feasible resource, a site-based reliability radius equation is also defined. The reliability of a computing site s_l can be computed as shown in equation (5.2.12).

$$\phi_l^{reliability} = \min_{\forall c_j^{allocated} \in s_l} \{\phi_j^{reliability}\} \tag{5.2.12}$$

The site-based reliability radius equation is shown in equation (5.2.13). The reliability radius of a site will decrease as more jobs are assigned to one single site. For instance, in event of a power failure occurring in an area where the computing site is, all resources at the site could become unavailable.

$$r_{\mu}^{site}(\phi^{reliability}, t) = \frac{\phi_l^{reliability}}{\sqrt{\text{number of applications allocated to computing site } s_l}} \tag{5.2.13}$$

where

$r_{\mu}^{site}(\phi^{reliability}, t)$ refers to the reliability radius of a given resource allocation, μ, of computing site s_l at time t.

The resource reliability metric $\rho_{\mu}^{resource}(\phi^{reliability}, t)$ of an allocation μ is the minimum reliability radii across all resources as shown in equation (5.2.14).

$$\rho_{\mu}^{resource}(\phi^{reliability}, t) = \min_{\forall j \in n} \{r_{\mu}^{resource}(\phi^{reliability}, t)\} \tag{5.2.14}$$

The site reliability metric of an allocation μ is the minimum reliability radii across all sites as shown in equation (5.2.15).

$$\rho_{\mu}^{site}(\phi^{reliability}, t) = \min_{\forall j \in n} \{r_{\mu}^{site}(\phi^{reliability}, t)\} \tag{5.2.15}$$

The goal is to maximise both reliability radii, resource and site, for a given set of resources. Maximising reliability radii will imply not only selecting the most reliable resources for the allocation but also a good distribution of forecasts on different reliable resources across sites. This will minimise the impact of a single site or resource failure and ensure that a maximum number of resources are successfully completed **(obligation 2)**.

5.3 Ensembles of Forecasts Allocation Patterns

Allocating resources to ensembles of forecasts is dependent on the number and size of resources available. The more and bigger the resources are, the more allocation options are available. More resources imply the possibility to choose the resources with the most optimal criteria, e.g. computing performance and reliability, for the forecasts. The bigger the resources, the higher the chances of finding sufficient cores/nodes for the allocated forecasts and also the possibility to have multiple concurrent forecasts on one resource.

In addition to allocating forecasts concurrently, there are also the possibilities to allocate forecasts consecutively and a combination of both, concurrently and consecutively. There are thus three main forecast allocation patterns as follows:

- Independent Consecutive Forecast Allocation Pattern
- Independent Concurrent Forecast Allocation Pattern
- Independent Concurrent and Consecutive Forecast Allocation Pattern

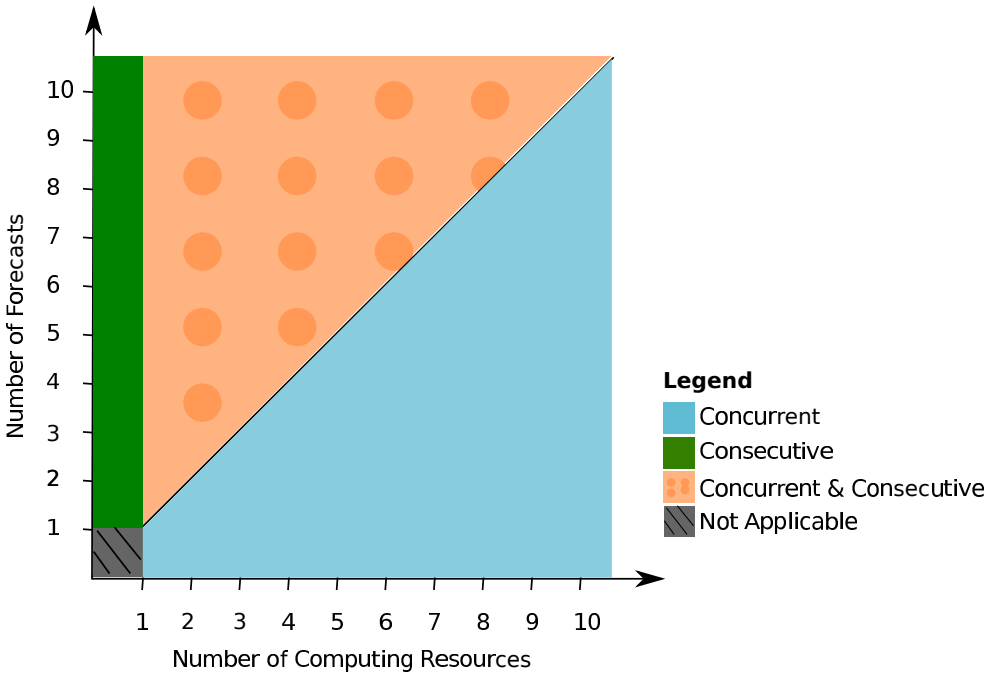


Figure 5.2: Area Chart of Forecast Allocation Patterns

Under the assumption that each resource has sufficient nodes/cores to compute only one forecast at any point in time, the recommended allocation patterns are illustrated in Figure 5.2. The gray area with stripes shows the case where there are only one forecast (not an ensemble) and is thus not applicable to us. The consecutive (green) area shows the scenario where there is only one resource but more than one forecast. The concurrent (blue) area illustrates the scenario where there are the same number of or more computing

resources as compared to the number of forecasts. Thus, all forecasts can compute concurrently. Consequently, all forecasts have to be computed consecutively. In the consecutive (orange with polka dots) area where there are more forecasts than the number of resources, forecasts have to be computed both concurrently and consecutively.

The pattern selected is dependent not only on the number of resources but also on the size of the resources. An example is one big resource where all forecasts can run concurrently. In this section, the possible cases for each allocation pattern will be elaborated. Depending on the pattern, the resulting execution time $f(\pi_j)$ on each resource c_j will be affected. Small size resources will refer to resources that can compute only one forecast at any point in time. Big size resources will represent resources that can compute more than one forecast at any point in time. It is assumed that all cores/nodes on a resource are available for urgent computations. Reallocation of applications or moving of applications from one resource to another upon failure will not be considered.

5.3.1 Independent Consecutive Forecast Allocation Pattern

The only possible combination for consecutive allocation pattern in terms of number and size of resources is represented by one case as follows.

Case 1: *One small* resource with sufficient nodes/cores to compute one forecast at any point in time. Forecasts thus are computed consecutively.

For the consecutive forecast allocation pattern, the execution time of the given resource is

$$f(\pi_1, i) = \sum_{i=1}^{n} \phi_{k1}(i) \tag{5.3.1}$$

where

i and n represent the instance and total number of applications assigned to the single resource c_1 respectively.

$\phi_{k1}(i)$ refers to the makespan of an application a_i on the resource c_1.

This is obviously the worst case forecast allocation pattern for ensemble of forecasts and should be avoided since makespan cannot be minimised. It also potentially presents a single point of failure.

5.3.2 Independent Concurrent Forecast Allocation Pattern

The possible combinations for concurrent allocation pattern in terms of number and size of resources can be summarised with four cases.

Case 1: *One big* resource with sufficient nodes/cores for all forecasts to compute concurrently.

Case 2: *Multiple small* resources where there are sufficient number of resources for each resource to compute one forecast.

Case 3: *Multiple small* and *big* resources where there are less number of resources as compared to the number of forecasts. Each small size resource has to compute one forecast while the big size resources will compute more than one forecasts concurrently.

Case 4: *Multiple big* resources where there are less number of resources as compared to the number of forecasts. The big size resources will compute more than one forecasts and all forecasts on all resources are computed concurrently.

For the concurrent forecast allocation pattern, the execution time of each given resource is

$$f(\pi_j, 1) = \phi_{kj}(1) \tag{5.3.2}$$

where

$\phi_{kj}(1)$ refers to the makespan of an application a_i ($i = 1$ since all applications are running concurrently) on resource c_j.

In the case where multiple forecasts are running currently, i.e. concurrent slots, on a big resource, each slot can be taken as if it is a single resource to simplify the equation. This is obviously the ideal forecast allocation pattern for ensembles of forecasts since it offers an allocation with the least makespan and should thus be adopted when possible. The main challenge is from finding resource providers and corresponding resources, both in numbers and sizes, to support and realise such an allocation. Case 1 should however be avoided since it can potentially result in a single point of failure.

5.3.3 Independent Concurrent and Consecutive Forecast Allocation Pattern

The possible combinations for concurrent and consecutive allocation pattern in terms of number and size of resources can be summarised with the three cases.

Case 1: *One large* resource with sufficient nodes/cores for more than one forecast but not all to compute concurrently. Forecasts that cannot compute concurrently are computed consecutively.

Case 2: *Multiple small* and *big* resources where there are less number of resources as compared to the number of forecasts. Each small size resource has to compute one forecast while the big size resources will compute more than one forecasts concurrently. The forecasts that are not computed concurrently are computed consecutively.

Case 3: *Multiple big* resources where there are less number of resources as compared to the number of forecasts. The big size resources will compute more than one forecasts concurrently. The forecasts that are not computed concurrently are computed consecutively.

For the concurrent consecutive forecast allocation pattern, the execution time of the given resource is

$$f(\pi_j, i) = \sum_{i=1}^{n} \phi_{kj}(i) \tag{5.3.3}$$

This is a realistic forecast allocation pattern when only a limited set of resources is available. Case 1 should be avoided due to the threat of a single point of failure.

Case Study 1 in Section 6.1 will evaluate the three forecast allocation patterns and demonstrate how makespan is influenced. The limitations of using only makespan robustness as an objective will also be shown.

5.4 Resource Allocation Heuristics

This section will elaborates the resource allocation heuristics that are developed to take advantage of the most ideal forecast allocation pattern, concurrent. The heuristics are as follows:

(i) Minimise Makespan (MM)

(ii) Minimise Makespan-Minimise ETS (MM-ME)

(iii) Minimise Makespan-Maximise Resource Reliability (MM-MRR)

(iv) Minimise Makespan-Maximise Resource Reliability-Minimise ETS (MM-MRR-ME)

(v) Minimise Makespan-Maximise Site Reliability (MM-MSR)

(vi) Minimise Makespan-Maximise Site and Resource Reliability-Minimise ETS (MM-MSRR-ME)

This set of six heuristics best illustrates how the objectives are incrementally fulfilled as illustrated in Figure 5.3. The (i) MM heuristic aims to only optimise the main objective, makespan robustness, and will serve as the basis to additionally fulfil of all other objectives.

(ii) MM-ME and (iii) MM-MRR will each realise one more objective, ETS and resource reliability, respectively. (iv) MM-MRR-ME combines the MM-ME and MM-MRR heuristics to manage makespan robustness, resource reliability and ETS. The objective, site reliability, is first included in the heuristic, (v) MM-MSR. The final heuristic, (vi) MM-MSRR-ME, aims to optimise all identified objectives by combining MM-MRR-ME and MM-MSR.

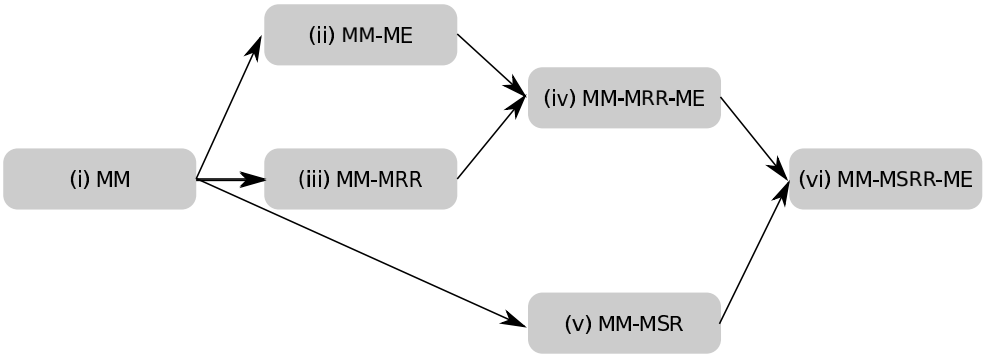


Figure 5.3: Evolution of Resource Allocation Heuristics

The priority for the objectives to be fulfilled is based on their importance to realise the requirements of urgent computing. Makespan is the most crucial objective since late results are useless as mitigation activities can no longer be carried out. Next, the reliability of an allocation is critical as the accuracy of the ensembles of forecasts will be adversely affected by the number of forecasts that fail to complete successfully. ETS is the least important objective as it is a constraint that if not optimised will only lead to energy wastage but will not directly influence the success and accuracy of forecasts.

The priority of the objectives is thus as follows:

(i) Robustness (makespan)

(ii) Reliability (site and/or resource)

(iii) ETS.

Table 5.1 shows the relationship between the heuristics and the objectives. The numbers in brackets illustrate the priority of the objectives in the respective heuristic. Empty cells imply that the corresponding objectives are not considered by the respective heuristics.

Case Study 2 in Section 6.2 evaluates these heuristics using a real flash flood use case and compares their performances based on the three objectives.

5.4.1 Minimise Makespan

The first heuristic, Minimise Makespan (MM), aims to maximise robustness and thus minimise the makespan of an allocation. The application requiring the longest mean execution time is assigned to the corresponding best performing computing resource.

Heuristic/Objectives	Makespan	Site Reliability	Resource Reliability	Energy-To-Solution (ETS)
MM	(1)			
MM-ME	(1)			(2)
MM-MRR	(1)		(2)	
MM-MRR-ME	(1)		(2)	(3)
MM-MSR	(1)	(2)		
MM-MSRR-ME	(1)	(2)	(3)	(4)

Table 5.1: Relationship between Resource Allocation Heuristics and their Objectives

The mean executive time χ for each application a_i on n computing resources can be calculated as follows:

$$\chi = \frac{\sum_{1}^{n} f(\pi_j)}{n} \qquad \text{and } j = 1..n \tag{5.4.1}$$

Consequently, the main rules for the MM heuristic are:

(i) Allocation order is based on the mean execution time χ of applications in descending order.

(ii) All applications are allocated to the fastest corresponding resources c_j at maximum frequency.

5.4.2 Minimise Makespan-Minimise ETS

This heuristic, Minimise Makespan-Minimise ETS (MM-ME), attempts to minimise both makespan (maximise makespan robustness) and ETS. The allocation rules are:

(i) Allocation order is based on the mean execution time χ of applications in descending order.

(ii) The application with the longest mean execution time χ is allocated to the fastest corresponding resource at maximum frequency.

(iii) All other applications will select a corresponding resource c_j that uses the least ETS, typically at a lower frequency, while ensuring that the execution time is less than the first allocated application.

5.4.3 Minimise Makespan-Maximise Resource Reliability

This heuristic, Minimise Makespan-Maximise Resource Reliability (MM-MRR), focuses on minimising makespan while maximising resource reliability. ETS is ignored. The allocation rules are:

(i) Allocation order is based on the mean execution time χ of applications in descending order.

(ii) The application with the longest mean execution time is allocated to the fastest corresponding resource at maximum frequency.

(iii) All other applications will select a corresponding resource c_j with a maximum resource reliability, while ensuring that the execution time is less than the first allocated application.

5.4.4 Minimise Makespan-Maximise Resource Reliability-Minimise ETS

This heuristic, Minimise Makespan-Maximise Resource Reliability-Minimise ETS (MM-MRR-ME), focuses on minimising makespan while maximising resource reliability and minimising ETS. The allocation rules are:

(i) Allocation order is based on the mean execution time χ of applications in descending order.

(ii) The application with the longest mean execution time is allocated to the fastest corresponding resource at maximum frequency.

(iii) All other applications will select a corresponding resource c_j with a maximum resource reliability with minimum ETS, while ensuring that the execution time is less than the first allocated application.

5.4.5 Minimise Makespan-Maximise Site Reliability

This heuristic, Minimise Makespan-Maximise Site Reliability (MM-MSR), will minimise makespan, and maximise site and resource reliability. ETS is ignored. The allocation rules are:

(i) Allocation order is based on the mean execution time χ of applications in descending order.

(ii) The application with the longest mean execution time is allocated to the fastest corresponding resource at maximum frequency.

(iii) All other applications will select a corresponding resource c_j with a maximum site reliability, while ensuring that the execution time is less than the first allocated application.

5.4.6 Minimise Makespan-Maximise Site and Resource Reliability-Minimise ETS

This final heuristic, Minimise Makespan-Maximise Site and Resource Reliability-Minimise ETS (MM-MSRR-ME), attempts to improve the previous MM-MRR heuristic by including site reliability, i.e. all objectives. The rules are as follow.:

(i) Allocation order is based on the mean execution time χ of applications in descending order.

(ii) The application with the longest mean execution time is allocated to the fastest corresponding resource at maximum frequency.

(iii) All other applications will select the corresponding resource(s) c_j with a maximum site reliability, while ensuring that the execution time is less than the first allocated application.

(iv) The resource with the maximum resource reliability and minimum ETS, i.e. lowest frequency, on c_j while maintaining the site reliability is selected for each application.

5.5 Assessment Model

An assessment model is introduced to provide a coherent framework to organise and interpret the results and metrics of resource allocations. It also facilitates the evaluations of different heuristics in meeting the defined obligations and objectives mentioned in Section 5.1 to enable comparison among different resource allocations for a given set of resources. In this model, all metrics are normalised to a range between zero and one to allow easier comparison. Zero implies the worst case while one implies the best case.

5.5.1 Obligation 1 – Meeting the Stipulated Deadline

To assess if **(obligation 1)** *Meeting the stipulated deadline* is met, the following assessment metric based on the signum (or sign) function is defined.

$$\theta_\mu(obligation^{deadline}) = sgn(\rho_\mu(\Phi, \pi_j)) \tag{5.5.1}$$

where

$\theta_\mu(obligation^{deadline})$ refers to the assessment metric for meeting a deadline for a resource allocation μ.

$sgn()$ represents the signum function.

The interpretation of the assessment metric is as follows:

- $\theta_\mu(obligation^{deadline}) < 0 \implies$ Stipulated deadline is not met.

- $\theta_\mu(obligation^{deadline}) \geq 0 \implies$ Stipulated deadline is met.

5.5.2 Obligation 2 – Maximising the Number of Successfully Allocated and Completed Forecasts

To assess if **(obligation 2)** *Maximising the number of successfully allocated and completed forecasts* is met, the following assessment metric based on the minimum-maximum normalisation [111] method is defined.

$$\theta_\mu(forecasts^{allocated}) = \frac{\text{number of allocated forecasts}}{n} \tag{5.5.2}$$

where

$\theta_\mu(forecasts^{allocated})$ refers to the assessment metric for the number of successfully allocated forecasts for a resource allocation μ.

n refers to the total number of forecasts that has to be allocated in the ensemble.

Equation (5.5.2) will normalise the fraction of forecasts that are allocated to a range between zero (worst case) and one (best case) to provide an easy way to assess the effectiveness of a resource allocation μ in terms of maximising the number of allocated forecasts. This however does not imply that all allocated forecasts will be successfully completed. The successful completion of these forecasts will be dependent on the reliability of the resources that are selected. Thus the number of successfully completed forecasts will be assessed via the reliability objective in Section 5.5.3.

5.5.3 Heuristic Objectives

To assess how well the three objectives (refer to Section 5.1) are met, the following assessment metric based on the minimum-maximum normalisation [111] method is defined.

For a given set of resources,

$$\theta_\mu(objective) = \frac{obj_\mu - obj_{min}}{obj_{max} - obj_{min}} \tag{5.5.3}$$

where

objective refers to the assessed heuristic objective of a particular resource allocation μ.

obj_μ represents the quantified value, e.g. makespan robustness metric, of the assessed objective of a particular resource allocation μ.

obj_{min} illustrates the lowest possible value of obj.

obj_{max} is the highest possible value of obj.

Equation (5.5.3) will normalise the quantified values to a range between zero and one to provide an easy way to assess the results across heuristics for a given set of resources.

In order to ensure that zero implies the worst base and one the best case, for objective values, e.g. energy usage, where the opposite is true (zero is the best case while one the worst case), the following equation has to be applied instead.

For a given set of resources,

$$\theta_{\mu}(objective) = 1 - \frac{obj_{\mu} - obj_{min}}{obj_{max} - obj_{min}} \tag{5.5.4}$$

Equation 5.5.4 will ensure that the assessment metric for objectives such as energy usage are normalised so that zero and one refer to the worst and best cases respectively.

The corresponding assessment metric for each defined objective is illustrated as follows:

Makespan Robustness

For a given set of resources,

$$\theta_{\mu}(\phi_k) = \frac{\rho_{\mu}(\Phi, \pi_j) - \rho(\Phi, \pi_j)_{min}}{\rho(\Phi, \pi_j)_{max} - \rho(\Phi, \pi_j)_{min}} \tag{5.5.5}$$

where

$\theta_{\mu}(\phi_k)$ refers to the makespan robustness assessment metric of a particular resource allocation μ.

$\rho_{\mu}(\Phi, \pi_j)$ is the makespan robustness metric value of a particular resource allocation μ.

$\rho(\Phi, \pi_j)_{min}$ illustrates the lowest possible value of makespan robustness metric.

$\rho(\Phi, \pi_j)_{max}$ is the highest possible value of makespan robustness metric.

Reliability

For a given set of resources,

$$\theta_{\mu}(\phi^{reliability}) = \frac{\theta_{\mu}^{resource}(\phi^{reliability}) + \theta_{\mu}^{site}(\phi_k)}{2} \tag{5.5.6}$$

where

$\theta_{\mu}(\phi^{reliability})$ refers to the overall reliability assessment metric of a particular resource allocation μ.

$\theta_{\mu}^{resource}(\phi^{reliability})$ refers to the resource reliability assessment metric of a particular resource allocation μ.

$\theta_{\mu}^{site}(\phi^{reliability})$ refers to the site reliability assessment metric of a particular resource allocation μ.

Resource and site reliability will be given equal weight in the assessment model. The resource and site assessment metrics are defined as follows:

(i) Resource Reliability

For a given set of resources,

$$\theta_{\mu}^{resource}(\phi^{reliability}) = \frac{\rho_{\mu}^{resource}(\phi^{reliability}, t) - \rho^{resource}(\phi^{reliability}, t)_{min}}{\rho^{resource}(\phi^{reliability}, t)_{max} - \rho^{resource}(\phi^{reliability}, t)_{min}} \tag{5.5.7}$$

where

$\rho_{\mu}^{resource}(\phi^{reliability}, t)$ represents the resource reliability metric value of a particular resource allocation μ.

$\rho^{resource}(\phi^{reliability}, t)_{min}$ illustrates the lowest possible value of resource reliability metric.

$\rho^{resource}(\phi^{reliability}, t)_{max}$ is the highest possible value of resource reliability metric.

(ii) Site Reliability

For a given set of resources,

$$\theta_{\mu}^{site}(\phi_k) = \frac{\rho_{\mu}^{site}(\phi^{reliability}, t) - \rho^{site}(\phi^{reliability}, t)_{min}}{\rho^{site}(\phi^{reliability}, t)_{max} - \rho^{site}(\phi^{reliability}, t)_{min}} \tag{5.5.8}$$

where

$\rho_{\mu}^{site}(\phi^{reliability}, t)$ represents the site reliability metric value of a particular resource allocation μ.

$\rho^{site}(\phi^{reliability}, t)_{min}$ illustrates the lowest possible value of site reliability metric.

$\rho^{site}(\phi^{reliability}, t)_{max}$ is the highest possible value of site reliability metric.

Energy Usage

For a given set of resources,

$$\theta_{\mu}(energy) = 1 - \frac{ets_{\mu} - ets_{min}}{ets_{max} - ets_{min}} \tag{5.5.9}$$

where

$\theta_{\mu}(energy)$ refers to the the energy usage assessment metric of a particular resource allocation μ.

ets_{μ} represents the ETS value of a particular resource allocation μ.

ets_{min} illustrates the lowest possible value of ETS.

ets_{max} is the highest possible value of ETS.

5.6 Summary

In this chapter, the research work leading to the set of resource allocation heuristics is presented. The two obligations, meeting the stipulated deadline and maximising the num-

ber of successfully allocated and completed forecasts, and three objectives, makespan robustness, site and resource reliability, and energy usage that the heuristics have to fulfil are identified.

In order to quantify these objectives, a robustness model and a reliability model are implemented. The most crucial timing constraints of urgent computing has to be fulfilled by minimising makespan, which is dependent on the selected forecast allocation patterns. Thus, the patterns are studied and three patterns, consecutive, concurrent and a combined concurrent and consecutive, are identified.

The appropriateness of a pattern is dependent on the number and size of resources available corresponding to the number of forecasts required and the number of nodes/cores each forecast needs. The concurrent pattern is the most ideal for minimising makespan as will be illustrated in Section 6.1.

Consequently, a set of resource allocation heuristics to optimise the three identified objectives in this order, makespan robustness, site and/or resource reliability and energy usage. This order is designed to reflect their importance in realising the requirements of urgent computing.

Finally, to assess and compare the results and metrics of the resource allocations, an assessment model is defined in terms of the identified obligations and objectives. The heuristics will be studied in Section 6.2 to understand the required compromises among the objectives. The results will be quantified with the robustness and reliability models and evaluated with the assessment model.

Chapter 6

Implementation and Result

Forecasting the onset and advancement of extreme events under time constraints requires reliable and robust models, which should take into account energy-aware allocations on HPC infrastructures. In this chapter, the forecast allocation patterns (Case Study 1), the robustness and reliability models, and the set of heuristics (Case Study 2) from Chapter 5 are investigated in two case studies.

(i) Case Study 1 – Gamma distribution

(ii) Case Study 2 – Flash flood

The case studies are selected to overcome the difficulties faced during the search for suitable use cases. Three use cases, earthquake, tsunami and flash flood, were originally identified as potential case studies. Ultimately, only one, flash flood, was adopted in this dissertation.

In the case of earthquakes, the seismologists with whom we are collaborating do not have a forecast model that can meet the stringent timing requirements of urgent computing and their domain science. Since earthquakes are in principle still considered by seismologists as unpredictable, the onset of earthquakes are only known when they have occurred, i.e. primary waves are detected by the seismic stations. However, the speed of the wave propagations are so fast that even well-known forecast models, e.g. SPECFEM3D Cartesian, are unable to compute fast enough to meet the stipulated deadline. The early earthquake warning system [42] is also not applicable for our collaborators in Italy. Due to their proximity to the fault, there is insufficient time to issue any warnings. Naturally, there are other possible use cases, e.g. predicting the aftershocks, which can potentially leverage on urgent computing. Unfortunately, they are outside the scope of the collaborating seismologists' field of expertise.

In the case of tsunamis, another ideal use case of urgent computing, we consulted the colleagues of our collaborators in Italy who are working on an early tsunami warning system, a candidate for the main Italian tsunami warning system. In this case, the scientists decided to adopt a Forecast Propagation Database for Tsunami [103]. This basically means that "all" expected possibilities of a tsunami are computed in advance and when the tsunami occurs, the best matched case is used to predict the advancement of the tsunami. This method is possible for tsunamis as there are many inherent uncertainties in the typically incomplete geophysical observational data and forecast models, making the limitations from using only best-matched pre-computations less pronounced. The application code, i.e. forecast model, used by this group of seismologists is private and not shared. Consequently, we do not have the domain science code and knowledge required to adopt this use case.

The flash flood use case that is provided by our collaborating hydro-meteorologist, perfectly demonstrates why urgent computing is necessary for disaster mitigation. However, we face the following limitations.

- Insufficient available computing resources
- Small set of forecasts in the given ensemble

In order to overcome the limitations, we introduce a Gamma distribution equation in Case Study 1 to generate experimental data for a set of thirty forecasts on ten resources. The bigger set of forecasts and resources is necessary to enable a variety of resource allocations. This will support the investigation on how the forecast allocation patterns will influence the makespan of an allocation and the effectiveness of a resource allocation with only one objective, makespan robustness. The usefulness of the makespan robustness metric in supporting the resource allocation is also studied. Other objectives, reliability and energy usage, are ignored in this case study.

The best allocation pattern, concurrent, for makespan that is identified in Case Study 1 is used to evaluate the set of heuristics to find the optimal allocation on distributed HPC resources for an ensemble of real hydro-meteorological, flash flood, forecasts in Case Study 2. The limited number of computing resources (two production HPC resources at one single site) is overcome by introducing three hypothetical resources. These hypothetical resources are necessary to simulate the required distribution of resources across sites and the different computing behaviours of the resources and forecasts. The ensemble of forecasts for a flash flood is allocated to this set of resources to compare and quantify the heuristics based on the obligations and objectives. The effectiveness of the proposed heuristics is evaluated and compared with the assessment model. The ultimate aim is to provide a robust and reliable resource allocation heuristic that can minimise ETS while meeting the deadline of an ensemble of forecasts in urgent computing.

6.1 Case Study 1 - Gamma distribution

Thirty ensembles of forecasts and ten resources will be available to this case study. The given $t_{deadline}$ is 3. The makespan of each forecast application on each resource is generated with a Gamma distribution as follows:

$$\phi_{kj}(i) = Gamma(shape, scale) * 0.15 + \beta \tag{6.1.1}$$

where

shape is set to $1 + \frac{i-1}{10}$, *scale* is set to 1.0 and β is set to $1 + \frac{j-1}{10}$.

Thus, all forecast applications on each given resource c_j will have the same β value while each forecast application i on all given resources will share the same *shape* value. This is to produce realistic experimental data where β will represent the execution time performance of a resource while *shape* will model the algorithmic execution time of a specific application. The gamma distribution will provide the randomness to model the compatibility between a resource and an application.

The resulting execution time (y-axis) distribution for each application (x-axis) is illustrated in Figure 6.1. Each symbol represents the execution time of a forecast application, A1 to A30 (corresponding to a_1 to a_{30}), on a given computing resource c_j (where $j = 1..10$). In general, the resources with a smaller j will have a comparatively better execution time performance.

Due to the computational complexity [108][p.636] involved in computing the robustness radius, near-optimal heuristic [65] approaches are typically employed. In the following subsections, the results are based on simple heuristic approaches.

6.1.1 Independent Consecutive Forecast Allocation Pattern

In the case of consecutive allocation pattern, there is only one single resource available (refer to Section 5.3.1). The execution time $f(\pi_1, i)$ (first row) and makespan robustness metric (second row) of the allocations on each of the ten resources is shown in Table 6.1. In principle, resource R1 (first column) has the best makespan robustness metric (-6.84). However since the value of the metric is negative, it means that the deadline is not met. In fact, there are no resources that can fulfil the deadline requirement since all $f(\pi_1, i) < 0$.

In a consecutive allocation pattern, the execution time of a resource is equivalent to the the sum of execution times of all forecasts as they are computed one after another on one resource. In such cases, best-effort allocations that attempt to maximise the number of allocated application forecasts are recommended.

Table 6.2 shows the best effort resource allocations on each resource in event that only one resource is available during the allocation. Each row represents an allocation μ when only that resource is available. On resources R1 to R5 (first to fifth row), two forecasts are allocated to each resource. The best makespan robustness metric for two allocated forecasts

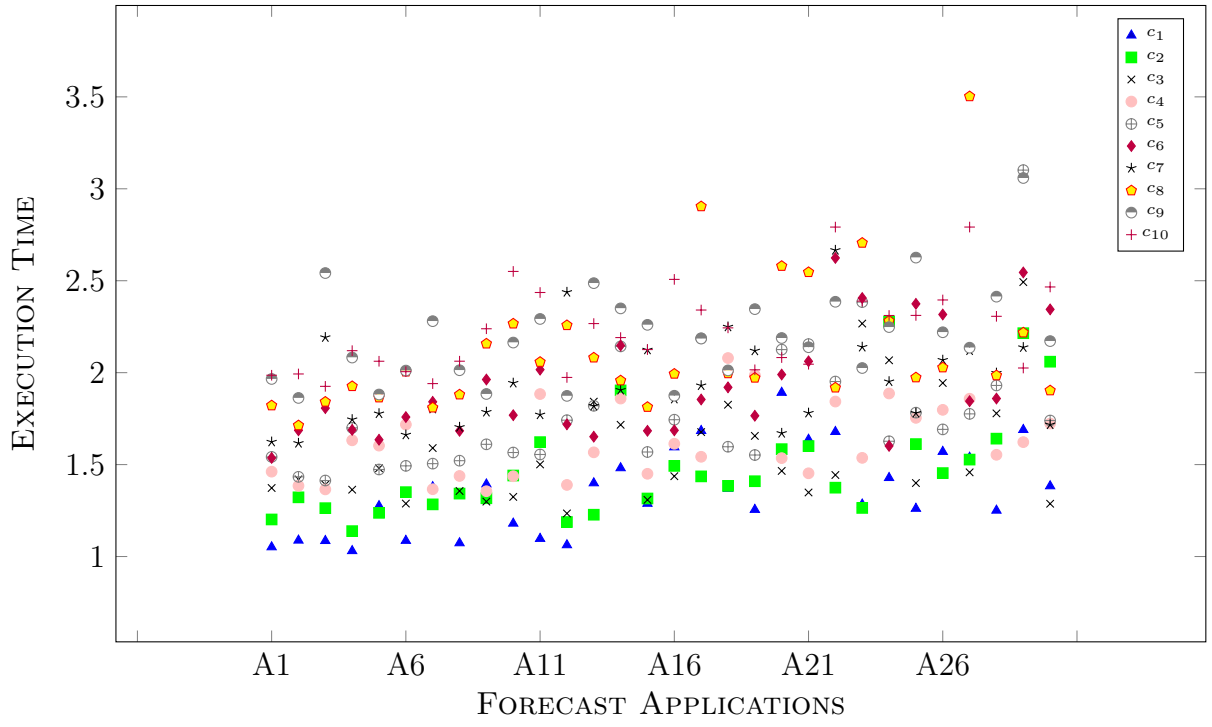


Figure 6.1: Gamma Distribution of Execution Times for 30 Ensembles Forecasts on each Resource

Resources/ Values	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**
$f(\pi_1, i)$	40.48	44.49	47.03	48.7	53.43	57.78	58.09	63.96	65.98	66.9
Robustness Metric	-6.84	-7.58	-8.04	-8.34	-9.21	-10.00	-10.06	-11.13	-11.50	-11.67

Table 6.1: Execution Time and Robustness Metric of a Thirty Forecast Applications Allocation for Each Computing Resource

is 0.65 on resource R1. This corresponds to the equation (6.1.1) and the findings from Table 6.1, where R1 is the resource with the best execution time performance. Additionally, there is an increasing trend in the magnitude of execution times (third column) and a decreasing trend (fourth column) in the robustness metrics from R1 to R5. This shows that the robustness metric is correctly illustrating that the smaller the execution time of a forecast, the more time there is left for tolerating perturbations in makespan.

Resources	**Allocated Forecast Applications**	**Execution time** $f(\pi_1, i)$	**Robustness Metric**
R1	A4, A1	2.08	0.65
R2	A4, A12	2.33	0.48
R3	A12, A30	2.52	0.34
R4	A9, A3	2.72	0.20
R5	A3, A2	2.85	0.11
R6	A1	1.54	1.46
R7	A2	1.62	1.38
R8	A2	1.71	1.29
R9	A2	1.86	1.14
R10	A3	1.93	1.07

Table 6.2: Consecutive Applications on Each Resource

Resources R6 to R10 are allocated only one forecast per resource within the timing constraint as shown in Table 6.2. Among these allocations, R6 has the best makespan robustness metric (1.46). Similar to the trends illustrated by R1 to R5, R6 to R10 show a decreasing tolerance for system perturbations as execution times on each resource increase. What is interesting to note is that the metric of R6 is the best among all resources. In fact, the metrics of R6 to R10 show significantly better makespan robustness performance when compared to R1 to R5. Since R1 to R5 have two allocated forecasts, the execution times are longer and thus their robustness metrics are reduced. However, they are in fact better allocations since they will potentially result in more forecasts being allocated and completed (one of the obligations of urgent computing) before the stipulated deadline.

In conclusion, consecutive forecast allocation pattern can significantly increase the execution times of an allocation. The makespan robustness metric is used to identify whether the deadline is missed by checking the sign of the metric. If the metric is negative, the deadline is missed. In cases where deadlines cannot be met, the maximum number of forecasts should be allocated. In such cases, the computed makespan robustness metric is insufficient to illustrate the effectiveness of an allocation since it cannot illustrate the number of successfully allocated forecasts.

6.1.2 Independent Concurrent Forecast Allocation Pattern

From equations (5.2.2) and (5.3.2), it can be derived that to maximise robustness, the forecast application with the longest execution time should be allocated to the fastest resource. Thus, the adopted order of allocation is to give priority to applications with higher mean execution times as in heuristic MM (refer to Section 5.4.1). Since ten resources are available in this case study, it will be assumed that each resource can accommodate all ensemble forecasts concurrently if required (Case 4 in Section 5.3.2).

Resources	**Allocated Forecast Applications**	**Execution Time $f(\pi_j, 1)$**	**Robustness Radius**
R1	A24, A14, A25, A28, A18, A11, A19, A10, A15, A12, A3, A4, A6, A8, A1, A2	1.48	1.52
R2	A22, A23, A17, A26, A13, A7, A5	1.45	1.55
R3	A27, A20, A30, A21, A16, A9	1.47	1.53
R4	A29	1.62	**1.38**

Table 6.3: Concurrent Applications on Multiple Resources

The result of an allocation μ using the MM heuristic is shown in Table 6.3 where the corresponding allocated applications, execution time and robustness radius of each resource are shared. All thirty ensemble forecasts are assigned only to the first four resources, R1 to R4, in order to minimise makespan by utilising the best execution time performance resource for each application. Since the robustness metric value is positive, the deadline is successfully met in this allocation. The makespan robustness metric of this allocation (last row) is 1.38 (highlighted in blue) with a makespan of 1.62. This metric value is better when compared to the consecutive allocations shown in Table 6.2. Thus, concurrent allocation pattern is beneficial to makespan robustness.

However the over-subscription of a few resources, four resources as shown in Table 6.3, can potentially lead to reliability issues, i.e. "a single point of failure". For example, if R1 fails, potentially 16 forecasts might not complete successfully.

Thus, another test (refer to Table 6.4) is conducted by adding a constraint of three concurrent applications per resource so as to provide a good distribution of forecasts across the ten resources using the MM heuristic. The resulting allocation has a lower robustness metric (0.99 highlighted in blue), and a longer makespan (2.01). Makespan is thus sacrificed for a more balanced load across all resources by allocating only three forecasts per resource. This implies that risk of a single resource failure will have less impact on the number of affected applications, i.e. three forecasts. Thus, this allocation is in fact better in terms of reliability.

In conclusion, the concurrent forecast allocation pattern successfully allocated all forecasts within the deadline while minimising the makespan such that a good tolerance to perturbations is possible. However, it does not show the actual distribution of forecasts among resources. Consequently, it can potentially create problems where the over-subscription of

Resources	Allocated Forecast Applications	Execution Time $f(\pi_j, 1)$	Robustness Radius
R1	A24, A14, A26	1.57	1.43
R2	A22, A23, A17	1.44	1.56
R3	A27, A20, A25	1.47	1.53
R4	A29, A21, A28	1.62	1.38
R5	A18, A11, A19	1.60	1.40
R6	A13, A16, A10	1.77	1.23
R7	A30, A9, A7	1.80	1.20
R8	A15, A3, A4	1.93	1.07
R9	A12, A5, A8	2.01	**0.99**
R10	A6, A1, A2	2.01	**0.99**

Table 6.4: Concurrent Applications on Multiple Resources with Constraint (3 Concurrent Application Per Resource)

some resources results in a "single point of failure". Nonetheless, a better distribution of the forecasts might come at the expense of makespan and makespan robustness, which is not preferred. Thus, a set of priority-based heuristics is developed as shown in Section 5.4.

6.1.3 Independent Concurrent and Consecutive Forecast Allocation Pattern

In the case of a concurrent and consecutive allocation pattern, we will assume a few initial constraints that are based on case 2 in Section 5.3.3. The constraints are:

(i) The better performing resources R1 to R5 each allow 3 concurrent applications.

(ii) Resources R6 to R10 each allow only 1 concurrent application.

(iii) Consecutive applications are allowed on all resources as long as $t_{deadline}$ is not violated.

Since the proposed heuristics, including MM, in Section 5.4 are meant for the preferred concurrent forecast pattern, a heuristic with reference to MM to evaluate a concurrent and consecutive allocation pattern is introduced as follows:

(i) Applications with the longest mean execution time are allocated first to available concurrent slots.

(ii) The remaining applications are matched with free consecutive slots on available resources while ensuring that the total execution times $f(\pi_j, i)$ are minimised and the robustness radii are maximised.

Resources (Concurrent Slot)	**Allocated Forecast Applications**	**Execution Time $f(\pi_j, i)$**	**Robustness Radius**
R1(1)	A24, A15	2.72	0.20
R1(2)	A14, A12	2.54	0.32
R1(3)	A26, A3	2.66	0.24
R2(1)	A22, A4	2.51	0.34
R2(2)	A23, A7	2.55	0.32
R2(3)	A17, A5	2.67	0.23
R3(1)	A27, A8	2.81	0.13
R3(2)	A20, A1	2.84	**0.11**
R3(3)	A25, A6	2.69	0.22
R4(1)	A29	1.62	1.38
R4(2)	A21, A2	2.84	**0.11**
R4(3)	A28	1.55	1.45
R5(1)	A18	1.60	1.40
R5(2)	A11	1.56	1.44
R5(3)	A19	1.55	1.45
R6	A13	1.65	1.35
R7	A30	1.72	1.28
R8	A10	2.27	0.73
R9	A16	1.87	1.13
R10	A9	2.24	0.76

Table 6.5: Concurrent and Consecutive Applications on Multiple Resources

Forecast Allocation Pattern	**Meeting Deadline** $\theta_\mu(obligation^{deadline})$	**Allocated Forecasts** $\theta_\mu(forecasts^{allocated})$
Consecutive	-1	0.03 or 0.07
Concurrent	1	1.0
Concurrent and Consecutive	1	1.0

Table 6.6: Assessment Metrics for Obligations

Table 6.5 shows the resulting allocation μ and the corresponding allocated applications, execution time and robustness radius of each resource. Each concurrent slot of a resource is represented as a single resource since the influence on makespan and robustness radius due to perturbations are per concurrent slot basis and not cumulative across slots. The robustness metric (minimum of the fourth column) of this allocation is 0.11 (highlighted in blue). There is a a good distribution of jobs across resources where each resource hosts at least one application. Resources R1, R2 and R3 are allocated 6 applications each, which can potentially result in reliability issues if they fail. In the case of R1 (first to third row), applications A24, A14 and A26 are computed concurrently. Applications A15, A12 and A3 are computed consecutively after A24, A14 and A26 respectively. The resources with more consecutive allocated applications have a significantly worse execution times (third column), 2.51 to 2.84, as compared to 1.55 to 2.27 on resources with only concurrent applications. Robustness radii (fourth column) are also significantly worse off, between 0.11 and 0.34, as compared to 0.73 to 1.45.

In conclusion, the concurrent and consecutive forecast allocation pattern successfully enables the allocation of all forecasts but resulted in a worse off robustness metric when compared to the concurrent forecast allocation pattern in the previous section. This shows that concurrent allocation pattern is the most effective allocation pattern for makespan when an ensemble of forecasts is allocated.

6.1.4 Assessment of Results

The assessment model defined in Section 5.5 for the two obligations can be applied to this case study as shown in Table 6.6. The consecutive forecast pattern (first row) neither meets the deadline obligation (deadline assessment metric is -1) nor maximises the number of allocated forecasts (allocated forecasts assessment metric < 1.0). If only one or two forecasts are allocated on the given resource, the allocated forecasts assessment metric is 0.03 or 0.06 respectively. The concurrent forecast pattern (second row), and concurrent and consecutive forecast pattern (third row) both manage to meet the stipulated deadline (deadline assessment metrics are 1) and maximise the number of allocated forecasts (allocated forecasts assessment metrics are 1.0). Thus, the concurrent forecast pattern, and concurrent and consecutive forecast pattern are suitable patterns to support ensembles of forecasts.

As the given set of resources and resource constraints for each resource pattern is different, e.g. one single resource and a constraint of three concurrent applications per resource, the assessment metric for makespan robustness will not be able to provide a fair comparison among the different allocations. However, the makespan robustness metric of each allocation offers an insight into the effectiveness of an allocation in minimising makespan. The smaller the metric, the better is the robustness towards makespan perturbations. The robustness model thus successfully helps to identify the most makespan effective forecast allocation pattern, concurrent, (makespan robustness metric is 1.38 or 0.99 from Table 6.3 and 6.4 respectively) to support ensembles of forecasts.

In summary, the case study shows that the most efficient forecast allocation pattern in terms of makespan robustness is the concurrent pattern. The robustness model is useful not only to support the allocation of a forecast to the best execution time performance computing resource, the metric is also useful to determine if the deadline is met (the sign of the metric) and the tolerance of the allocation towards makespan perturbations (the magnitude of the metric).

Additionally, the results of the case study highlight the limitations of using only makespan robustness as an objective. Although makespan is successfully minimised, an unbalanced distribution of forecasts among the available resources can occur. In some allocations, most forecasts are allocated to a few resources with good execution time performances. This potentially increases the risk of a "single point of failure" when one of such resources fails.

There is thus a need to include other objectives, e.g. reliability, to improve the effectiveness of an allocation. However, these additional objectives should not affect makespan, the most crucial objective. Concurrent forecast allocation pattern should also be the adopted pattern to minimise makespan.

In the next case study, a set of priority-based heuristics, which will incrementally adopt the additional objectives on a set of resources where concurrent forecast allocation pattern is possible, is studied. The effect of introducing the additional objectives will be evaluated to ensure that makespan is not adversely affected while ensuring all objectives are optimised.

6.2 Case Study 2 - Flash flood

The WRF-ARW (Weather Research & Forecasting-Advanced Research WRF) Model[1] will be used to forecast the Genoa flash flood that occurred on 9th October 2014 where no advance warning was given, and one death and a cost of 303 million USD[2] ensued. Each computation will use 640 cores to generate a 24 hours forecast with two domains, 5 and 1 km, within 3 hours, the stipulated deterministic deadline. An ensemble of forecasts with eight microphysics options, Kessler, WSM3, Ferrier, WSM5, WSM6, WSM6D,

[1] http://www.wrf-model.org/
[2] http://www.emdat.be/

Thompson and Morrison, have to be allocated to a set of predefined resources. Each microphysics option is a different representation of atmospheric heat and moisture tendencies, microphysical rates and surface rainfall [112].

The evaluation of the effectiveness of the resource allocation will be performed with two production petaflops HPC resources, SuperMUC and SuperMUC Phase 2[3], hosted at Leibniz Supercomputing Centre (LRZ) and three hypothetical resources. All data from the production resources are collected from real experiments.

The targeted HPC resources are shown in Table 6.7 with information about their respective available cores, default frequency and availability. The hypothetical resources are included to measure the proposed heuristics in terms of multiple sites, high and low reliability, etc., which are otherwise not possible since both real HPC resources are hosted at one site with similar reliability values. Both SuperMUC and SuperMUC Phase 2 have sufficient available cores (second column) such that all forecasts can potentially compute concurrently on them. Resources RA, RB and RC have less available cores and can each only accommodate three, four and seven concurrent forecasts respectively. The default frequency (third column) of each resource is also shared. To introduce a variety of sites (fourth column), Resource RA is assumed to be hosted at Site1 while resources RB and RC are assumed to be hosted at Site2. SuperMUC and SuperMUC Phase 2 are hosted at LRZ. The availability (fifth column) of the resources is such that RA is the worst, follow by RB, then SuperMUC and SuperMUC Phase 2, and finally RC with the best availability.

Table 6.7: Targeted HPC Resources

Resource	Available cores	Default Freq.(GHz)	Site	Availability
SuperMUC	147,456	2.3	LRZ	0.85
SuperMUC Phase 2	86,016	2.2	LRZ	0.85
RA	3,000	2.6	Site1	0.70
RB	2,000	2.6	Site2	0.80
RC	5,000	2.5	Site2	0.90

Figure 6.2 shows the predicted execution time and ETS of SuperMUC, SuperMUC Phase 2, and the hypothetical resources for eight microphysics options, i.e. eight different forecasts. It represents the intitial condition of this case study. The execution times (first and second chart in the first row) and ETS (first and second chart in the second row) readings for SuperMUC and SuperMUC Phase 2, are collected at the default frequency, i.e. 2.3 and 2.2 GHz respectively. The estimated execution time and ETS of the scaled-up frequencies are computed with the LoadLeveler Prediction Model [110]. The available frequencies for SuperMUC and SuperMUC Phase 2 are 2.3 to 2.7 GHz and 2.2 to 2.6 GHz respectively. The execution time and ETS of hypothetical resources are correspondingly chosen to serve as a contrast to the production resources. RA is the best performing resource (third chart in the first row) in terms of execution time, follow by RB and then

[3]https://www.lrz.de/services/compute/supermuc/systemdescription/

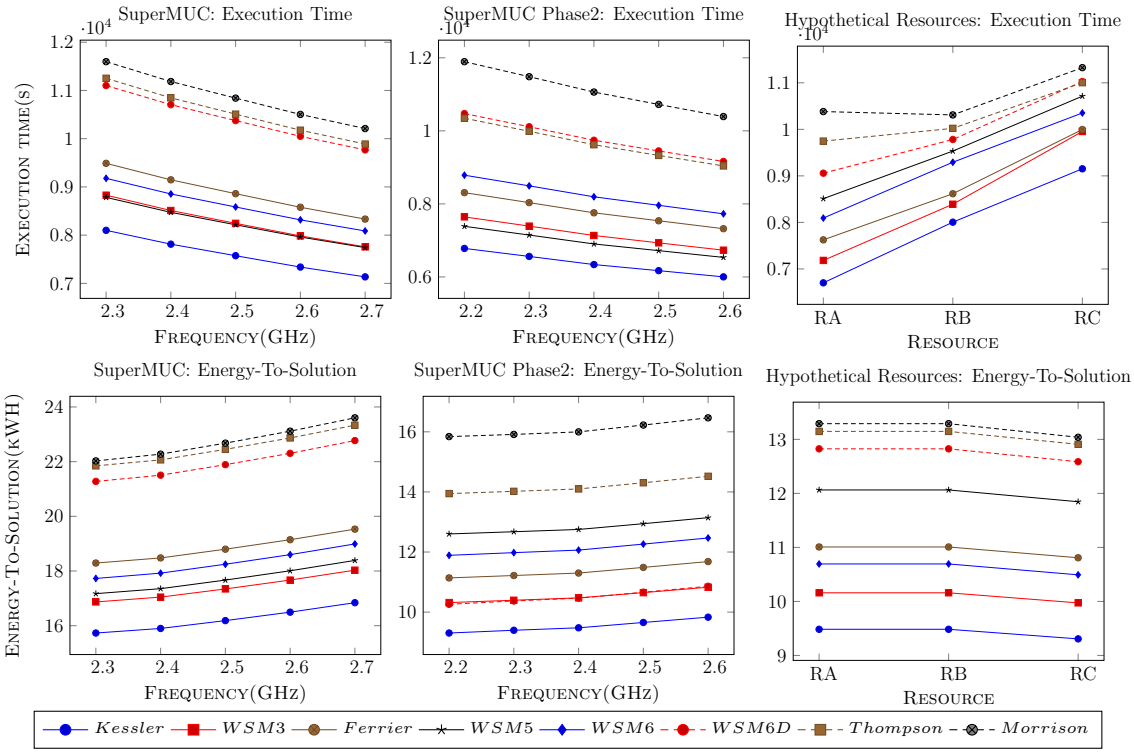


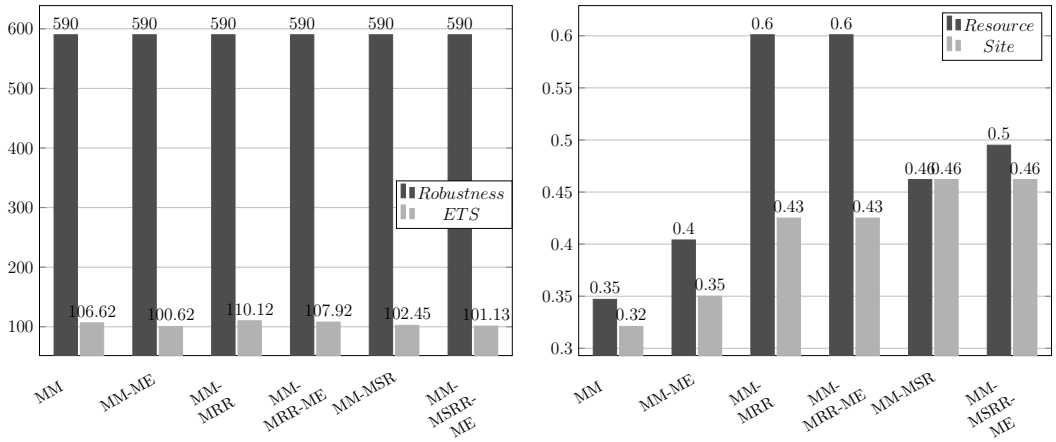
Figure 6.2: Execution Time and ETS Data

RA. Resources RA, RB and RC will not offer frequency scaling and thus the respective ETS (third chart in the second row) is based on their default frequencies. RA and RB have similar ETS while RC requires the least ETS.

Evaluation of the six heuristics, MM, MM-ME, MM-MRR, MM-MRR-ME, MM-MSR and MM-MSRR-ME, is performed by allocating eight flash flood ensemble forecasts to the above mentioned resource set. Figure 6.3a and 6.3b show the summarised quantified performance, robustness metric and ETS, and site and resource reliability metrics of the each heuristic respectively. All heuristics aim to maximise robustness by minimising makespan, i.e. ensuring that the forecast with the longest mean execution time is the first to be assigned to the corresponding best execution time performance resource at maximum frequency, as the first priority. The allocation details of each heuristic and analysis of the results will be elaborated in the following subsections with the data from Figure 6.3a and 6.3b.

6.2.1 Minimise Makespan

Table 6.8 illustrates the selected allocation with MM heuristic. It is a naive heuristic that aims to minimise the makespan (10210 from the fifth column) by selecting the best performing resource and frequency for each forecast. All forecasts except Morrison and



(a) Robustness Metric and ETS (kWH) (b) Reliability Metric (Resource and Site)

Figure 6.3: Makespan Robustness, ETS, and Resource and Site Reliability Measurements

WSM6D are allocated to SuperMUC Phase 2 at maximum frequency of 2.6 GHz, leading to the over-reliance of SuperMUC Phase 2. Morrison is assigned to SuperMUC at its maximum frequency, 2.7 GHz, while WSM6D is allocated to RA at 2.6 GHz. This implies only resources from two sites (second column), LRZ and Site1, are utilised.

Figure 6.3a and 6.3b show that although robustness metric (590) is good, reliability (site reliability= 3.2 and resource reliability= 0.4) and ETS (106.62) are withheld. MM will have the least average execution time among all heuristics and is arguably the best heuristic for urgent use cases with non-deterministic deadline. In non-deterministic deadline cases, completing the maximum number of forecasts as soon as possible will be the most crucial criterion since deadlines are unknown. Consequently, constraints like reliability becomes less important and ETS can be ignored. In conclusion, MM manages to minimise makespan of an allocation by allocating each forecast to its corresponding best execution time performance resource.

Forecast	**Site**	**Resource**	**Frequency (GHz)**	**Execution Time (s)**	**ETS (kWh)**
Morrison	LRZ	SuperMUC	2.7	10210	23.602
Thompson	LRZ	SuperMUC Phase 2	2.6	9093	14.520
WSM6D	Site1	RA	2.6	9058	12.825
Ferrier	LRZ	SuperMUC Phase 2	2.6	7321	11.682
WSM6	LRZ	SuperMUC Phase 2	2.6	7728	12.467
WSM3	LRZ	SuperMUC Phase 2	2.6	7733	10.829
WSM5	LRZ	SuperMUC Phase 2	2.6	6535	10.860
Kessler	LRZ	SuperMUC Phase 2	2.6	6002	9.832

Table 6.8: Minimise Makespan Resource Allocation

6.2.2 Minimise Makespan-Minimise ETS

Table 6.9 illustrates the selected allocation with MM-ME heuristic. It is an energy efficient version of MM where the objectives are to minimise both makespan (10210 from fifth column) and ETS (sixth column). Forecasts Thompson, Ferrier, WSM6 and WSM3 are allocated to resources RA and RC instead of SuperMUC Phase 2 when compared to MM. WSM5 and Kessler are still allocated to SuperMUC Phase 2 but use a lower frequency, 2.2 GHz. Consequently, four resources from all three computing sites are utilised.

Figure 6.3a and 6.3b show that the makespan robustness (590) of MM-ME remains unchanged while the ETS (100.62) is reduced by 6 kWh when compared to MM. This is achieved by selecting resources that require less ETS and/or the same resource at a lower frequency. Since the makespan is determined by the forecast with the maximum execution time as all forecasts are running concurrently, there is in fact no need for forecasts with a shorter execution times than the first allocated forecast (forecast with the longest execution time in the allocation) to be allocated to the most efficient or expensive (in terms of ETS) resource or at the highest frequency. Thus it is possible to reduce ETS at no expense to makespan. The selection of more sites and resources led to an improvement in both site (0.4) and resource reliability (0.35) when compared to MM. In conclusion, MM-ME manages to reduce energy usage while ensuring that makespan remains unaffected.

Forecast	**Site**	**Resource**	**Frequency (GHz)**	**Execution Time (s)**	**ETS (kWh)**
Morrison	LRZ	SuperMUC	2.7	10210	23.602
Thompson	Site1	RA	2.6	9747	13.148
WSM6D	Site1	RA	2.6	9058	12.825
Ferrier	Site2	RC	2.5	9996	10.809
WSM6	Site1	RA	2.6	8093	10.694
WSM3	Site2	RC	2.5	9954	9.973
WSM5	LRZ	SuperMUC Phase 2	2.2	7386	10.261
Kessler	LRZ	SuperMUC Phase 2	2.2	6781	9.304

Table 6.9: Minimise Makespan-Minimise ETS Resource Allocation

6.2.3 Minimise Makespan-Maximise Resource Reliability

Table 6.10 illustrates the selected allocation with MM-MRR heuristic. It is a reliable version of MM where the objectives are to minimise makespan and maximise resource reliability. Forecasts WSM6D, Ferrier, WSM6, WSM3 and Kessler are allocated to different resources when compared to MM. The forecasts, Morrison, Thompson, WSM5 and Kessler, are allocated to the resources at LRZ at the maximum frequency since energy is not an objective in this heuristic. All five resources are utilised as opposed to only three in the case of MM. The forecasts have a better distribution on each resource and among the

resources. Each resource has only one to two allocated forecasts while each site has one to four allocated forecasts. The site LRZ is allocated four forecasts. If LRZ becomes unavailable due to unforeseen circumstances, e.g. power outage, half of the forecasts will not complete. Thus, in addition to resource reliability, site reliability is also important.

Figure 6.3a and 6.3b show that the makespan robustness (590) remains unchanged while the resource reliability metric is improved from 0.35 to 0.6 when compared to MM. This implies that the risk of many forecasts being affected by a single resource failure is reduced as compared to MM. The improvement in site reliability is attributed to a better distribution of forecasts among resources, which leads to resources from all sites to be chosen as compared to only resources from two sites in the case of MM. However, the reliability improvement comes at an expense. ETS (110.12) is increased by 3.5 kWh when compared to MM. In conclusion, MM-MRR manages to improve resource reliability while ensuring that makespan remains unaffected.

Forecast	Site	Resource	Frequency (GHz)	Execution Time (s)	ETS (kWh)
Morrison	LRZ	SuperMUC	2.7	10210	23.602
Thompson	LRZ	SuperMUC Phase 2	2.6	9093	14.520
WSM6D	Site2	RB	2.6	9782	12.825
Ferrier	Site2	RC	2.5	9996	10.809
WSM6	Site1	RA	2.6	8093	10.694
WSM3	Site2	RC	2.5	9954	9.973
WSM5	LRZ	SuperMUC Phase 2	2.6	6535	10.860
Kessler	LRZ	SuperMUC	2.7	7135	16.840

Table 6.10: Minimise Makespan-Maximise Resource Reliability Resource Allocation

6.2.4 Minimise Makespan-Maximise Resource Reliability-Minimise ETS

Table 6.11 illustrates the selected allocation with MM-MRR-ME heuristic. It is an improved version of MM-MRR where the objectives are to minimise makespan, maximise resource reliability and minimise ETS. There is no change in the allocated resources and sites when compared to MM-MRR. The main difference is the frequency choice of forecasts, Thompson, WSM5 and Kessler, which are allocated to SuperMUC or SuperMUC Phase 2. Reduced frequencies, 2.3 GHz, 2.2 GHz and 2.3 GHz for Thompson, WSM5 and Kessler respectively, are selected to improve ETS.

Figure 6.3a and 6.3b show that the makespan robustness (590), resource reliability (0.6) and site reliability (0.43) metrics remain unchanged but ETS (107.92) is reduced by 2.2 kWh when compared MM-MRR. ETS improvements are achieved by selecting a lower frequency when appropriate. However, this ETS value is still higher when compared to MM (refer to Table 6.8), which mainly utilises the more energy efficient resource, SuperMUC

Phase 2. In conclusion, MM-MRR-ME manages to reduce ETS while ensuring that resource reliability and makespan robustness remain unaffected.

Forecast	Site	Resource	Frequency (GHz)	Execution Time (s)	ETS (kWh)
Morrison	LRZ	SuperMUC	2.7	10210	23.602
Thompson	LRZ	SuperMUC Phase 2	2.3	9988	14.02
WSM6D	Site2	RB	2.6	9782	12.825
Ferrier	Site2	RC	2.5	9996	10.809
WSM6	Site1	RA	2.6	8093	10.694
WSM3	Site2	RC	2.5	9954	9.973
WSM5	LRZ	SuperMUC Phase 2	2.2	7386	10.261
Kessler	LRZ	SuperMUC	2.3	8100	15.732

Table 6.11: Minimise Makespan-Maximise Resource Reliability-Minimise ETS Resource Allocation

6.2.5 Minimise Makespan-Maximise Site Reliability

Table 6.12 illustrates the selected allocation with MM-MSR heuristic. It is a reliable version of MM where the objectives are to minimise makespan and maximise site reliability. It differs from MM-MRR as it measures the reliability based on site as opposed to resource. It attempts to minimise the risk of multiple forecasts not completing in event of a single site failure, e.g. a power outage. All three sites are thus allocated with two to three forecasts each. This leads to a better site distribution when compared to MM-MRR (refer to Table 6.10) where LRZ is allocated four forecasts and Site1 is allocated only one forecast. All five resources are utilised where each resource is allocated between one to three resources each. This is a less effective resource distribution when compared to MM-MRR where each resource has only one to two allocated forecasts each. If resource RB fails, three forecasts will not complete. Thus, both site reliability and resource reliability are equally important. As ETS is not an objective in this heuristic, forecasts that are allocated on SuperMUC and SuperMUC Phase 2, i.e. resources that support frequency scaling, compute with the maximum frequency of 2.7 GHz and 2.6 GHz respectively.

Figure 6.3a and 6.3b show that the makespan robustness (590) remains unchanged but site reliability (0.46) is improved as compared to both MM (0.32) and MM-MRR (0.43). The search for more reliable sites result in more energy efficient resources, a consequence of the initial condition of the case study, to be selected and thus a lower ETS (102.45) when compared to both MM (106.62) and MM-MRR (107.92). However, resource reliability is reduced to 0.46 when compared to 0.6 of MM-MRR. Optimising site reliability does not implicitly imply optimising resource reliability. In conclusion, MM-MSR manages to improve site reliability while ensuring that makespan remains unaffected.

Forecast	Site	Resource	Frequency (GHz)	Execution Time (s)	ETS (kWh)
Morrison	LRZ	SuperMUC	2.7	10210	23.602
Thompson	Site2	RB	2.6	10021	13.148
WSM6D	Site1	RA	2.6	9058	12.825
Ferrier	LRZ	SuperMUC Phase 2	2.6	7321	11.682
WSM6	Site2	RB	2.6	9293	10.694
WSM3	Site1	RA	2.6	7183	10.16
WSM5	LRZ	SuperMUC Phase 2	2.6	6535	10.86
Kessler	Site2	RB	2.6	8005	9.484

Table 6.12: Minimise Makespan-Maximise Site Reliability Resource Allocation

6.2.6 Minimise Makespan-Maximise Site and Resource Reliability-Minimise ETS

Table 6.13 illustrates the selected allocation with MM-MSRR-ME heuristic. It is arguably the most optimal heuristic for deterministic deadlines among this set of heuristics. Its objectives are to minimise makespan, maximise site and resource reliability, and minimise ETS. Resources from all three sites are selected where each site is allocated with two to three forecasts. Each resource has one to two allocated forecasts. This allocation offers the best distribution of forecasts among both resources and sites. Since energy usage is one of the objectives, the two forecasts, Ferrier and WSM5, which are assigned on SuperMUC Phase 2, use the default frequency of 2.2 GHz to reduce energy usage.

Figure 6.3a and 6.3b show that the makespan robustness (590) remains unchanged when compared to all other heuristics. Both resource reliability (0.5) and ETS (101.13) are improved at no expense to site reliability (0.46) when compared to MM and MM-MSR. Site reliability is improved or remains unchanged as compared to MM and MM-MSR respectively. However, in comparison to MM-MRR, the resource reliability is reduced to improve site reliability. ETS is the least in the heuristic set, with exception of MM-ME, which has a lower ETS. The lower ETS can be seen as the required compromise to improve site and resource reliability. MM-MSRR-ME has the best site reliability, and the second best resource reliability and ETS among the heuristics. In conclusion, MM-MSRR-ME manages to improve site and resource reliability, and ETS while ensuring that makespan remains unchanged.

6.2.7 Assessment of Results

In order to have a better overview of the performance of each heuristic in terms of the obligations and objectives defined, the assessment model defined in Section 5.5 is applied to the results of the above heuristics. The maximum and minimum values of each objective are computed from the given resource set, i.e. the two production resources and three

Forecast	Site	Resource	Frequency (GHz)	Execution Time (s)	ETS (kWh)
Morrison	LRZ	SuperMUC	2.7	10210	23.602
Thompson	Site2	RB	2.6	10021	13.148
WSM6D	Site1	RA	2.6	9058	12.825
Ferrier	LRZ	SuperMUC Phase 2	2.2	8308	11.138
WSM6	Site2	RB	2.6	9293	10.694
WSM3	Site1	RA	2.6	7183	10.160
WSM5	LRZ	SuperMUC Phase 2	2.2	7386	10.261
Kessler	Site2	RC	2.5	9152	9.307

Table 6.13: Minimise Makespan-Maximise Site and Resource Reliability-Minimise ETS Resource Allocation

Heuristics	Meeting Deadline $\theta_\mu(obligation^{deadline})$	Allocated Forecasts $\theta_\mu(forecasts^{allocated})$	Makespan Robustness $\theta_\mu(\phi_k)$	Reliability $\theta_\mu(\phi^{reliability})$	Energy $\theta_\mu(energy)$
MM	1.0	1.0	1.0	0.19	0.76
MM-ME	1.0	1.0	1.0	0.28	0.84
MM-MRR	1.0	1.0	1.0	0.92	0.71
MM-MRR-ME	1.0	1.0	1.0	0.92	0.74
MM-MSR	1.0	1.0	1.0	0.68	0.82
MM-MSRR-ME	1.0	1.0	1.0	0.75	0.84

Table 6.14: Assessment Metrics of the Heuristics

hypothetical resources, by ignoring all other objectives (refer to Appendix C).

The resulting assessment metrics of the two obligations (second and third columns) and three objectives (fourth to sixth columns) are shown in Table 6.14. All heuristics manage to fulfil the two obligations and makespan robustness objective with the best possible results (1). This implies that the stipulated deadline is met, all forecasts are successfully allocated and makespan robustness is maximised. Thus, all six heuristics are effective for allocating ensembles of forecasts on a given set of resources since the obligations are fulfilled and the first priority objective is maximised.

The optimisation of the two objectives, reliability and energy usage, is however dependent on the chosen heuristic. From Table 6.14, reliability is illustrated in the fifth column. MM has the worst reliability assessment metric (0.19 from the first row) while MM-MRR and MM-MRR-ME has the best reliability assessment metric (0.92 from the third and fourth row) among this set of heuristics. For energy usage (last column), MM-ME and MM-MSRR-ME offer the best energy assessment metric (third and last row) of approximately 0.84 while MM-MRR (third row) has the worst (0.71).

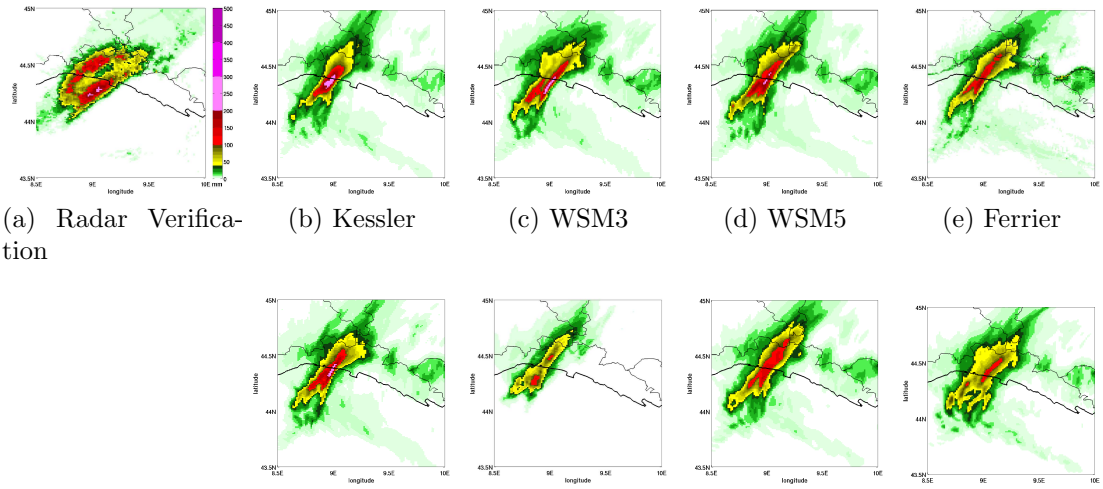
In summary, with the priority based resource allocation approach, all heuristics manage to realise the obligations and the makespan robustness objective with the best possible results. This implies that all heuristics can be used to enable urgent computing for an ensemble of forecasts. However, to realise the other two objectives, reliability and energy

usage, compromises among these two objectives are necessary. Depending on the computing scenario faced, the most appropriate heuristic can be used. The assessment results show that MM-MSRR-ME heuristic is the most optimal heuristic for fulfilling all three objectives as it manages to optimise each objective with minimum adverse influence to the other objectives. It is thus the recommended heuristic to enable a robust, reliable and energy-aware resource allocation.

6.2.8 Visualisation of Ensemble of Flash Flood Forecasts

The visualised results of the ensemble of forecasts at the 12th hour and 24th hour after the event occurred are shown in Figure 6.4 and 6.5 respectively. The different colours indicate the different amount of rainfall where the amount increases from white (0 mm) to violet (500 mm) (refer to the colour bar on the right of Figure 6.4a and 6.5a) . Accordingly, this ensemble is a reasonably accurate forecast, where three ensemble members, WSM3, WSM5 and WSM6, provide relevant results for early warning.

However, there is no single forecast from Figure 6.4b to Figure 6.4i that reflects the actual state of the event as shown in Figure 6.4a. In fact, by the 24th hour (Figure 6.5), all models, Figure 6.5b to Figure 6.5i, show a significant underestimation in the severity of the event when compared to the actual state (Figure 6.5a). This highlights the limitations of forecast models and collected data. It is thus crucial in such cases to involve the event domain scientist when evaluating the urgent products so that educated decisions can be made for mitigation activities.

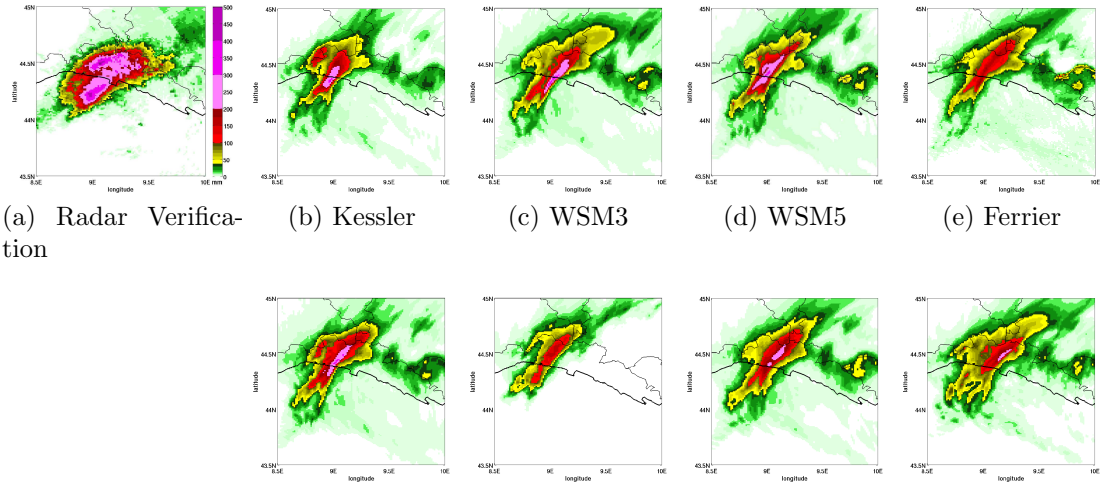


(a) Radar Verification (b) Kessler (c) WSM3 (d) WSM5 (e) Ferrier

(f) WSM6 (g) WSM6D (h) Thompson (i) Morrison

Figure 6.4: Ensemble of Forecasts at 12:00:00 UTC

In summary, the visualised findings of the ensemble of forecasts illustrate that there is no single deterministic method to generate a high fidelity forecast. Ensembles of forecasts are more useful in such cases to enable probabilistic forecasting. The results from an



(a) Radar Verification (b) Kessler (c) WSM3 (d) WSM5 (e) Ferrier

(f) WSM6 (g) WSM6D (h) Thompson (i) Morrison

Figure 6.5: Ensemble of Forecasts at 24:00:00 UTC

ensemble is expected to provide crucial inputs to facilitate decision support for an urgent event.

Chapter 7

Conclusion and Future Work

This dissertation explores how to effectively enable urgent computing by allocating ensembles of forecasts on multiple distributed heterogeneous computing resources before a stipulated deadline. The research problem originates from the timing constraints of urgent computing when using zero hour data to improve prediction accuracy and the absence of a single deterministic forecast model in many event domain sciences to generate a high fidelity forecast. Consequently, supporting ensembles of forecasts (stochastic method) utilising zero hour data to produce multiple forecasts of the the same events for probabilistic forecasting before the stipulated deadlines is the main research goal of this work. The ensemble forecast results will facilitate decision support for loss mitigation.

The preferred targeted distributed resources are public IT infrastructures, e.g. supercomputers and grids, since they offer an array of distributed computing resources that are the most cost effective to compute ensembles of forecasts. Alternatively, private and/or public clouds can also be utilised. However, coordinating multiple forecasts simultaneously under time constraints on numerous distributed computing resources is not trivial. This dissertation sought to address the challenge as follows:

(i) Provide an urgent computing framework to coordinate multiple forecasts, and resources and their environments simultaneously at zero hour from ubiquitous clients in view of the chaotic environments that typically accompany urgent events.

(ii) Adopt a heuristic approach within the urgent computing framework to provide near-optimal allocation of ensembles of forecasts to given sets of computing resources within the deadline constraints while adhering to the defined obligations and objectives (refer to Section 5.1).

In conclusion, the research goal is successfully achieved with the urgent computing framework and the heuristic approach. The research findings are summarised in Section 7.1. Potential future work is shared in Section 7.2.

7.1 Summary of Findings

The research work began with a combined quantitative and qualitative research methodology to investigate the characteristics and requirements of urgent computing and the event domain sciences. This methodology successfully helps to identify the then prevalent incomplete definition of urgent computing and the limited focus on the very real and crucial challenge, an absence of a deterministic high fidelity forecast model, faced by event domain scientists.

To tackle the incomplete definition, an in-depth analysis of the only definition of urgent computing was conducted. The shortcomings in the definition were identified. Related research activities were also revisited to gain a deeper understanding of the characteristics and requirements of urgent computing. This eventually led to a first comprehensive definition where urgent computing specific challenges are clarified. This new definition helps to demystify, in particular, the relationship of urgent computing and real-time computing where despite having many similarities, in particular the time constraints, they are not the same. The existence of non-deterministic deadlines and the need to still expect losses upon meeting deadlines are two distinctive urgent computing characteristics. The new urgent computing definition thus helps to align the focus of this dissertation to tackling the most crucial urgent computing challenges.

The absence of a single high fidelity deterministic forecast model in many event domain sciences resulted in a prevalence in stochastic methods where ensembles of forecasts are leveraged on to perform probabilistic forecasting. In order to support ensembles of forecasts, there is a need to confront the challenge to coordinate multiple forecasts on multiple distributed heterogeneous computing resources under time constraints. Despite the expected challenge, this event domain science requirement is considered as paramount and thus cannot be ignored. Our work became the first in urgent computing to target ensembles of forecasts.

Supporting ensembles of forecasts increase the challenge to design the urgent computing framework. A Task-based Ubiquitous (TbU) approach is consequently designed to realise the framework by providing ubiquitous access to manage multiple forecasts and resources from anywhere anytime. Ubiquitous access is considered as crucial in view of the chaotic environments that typically accompany an urgent event. The task-based feature provides the adaptability to refine the forecast models when additional insights are gained from zero hour data. A good indicator of the success of the TbU approach is the urgent computing visualisation service (refer to Appendix A), which was realised with the TbU approach. This service has since been adopted as a production service at LRZ.

The heuristic approach was additionally adopted within the urgent computing framework to support near-optimal allocation of ensembles of forecasts to given sets of computing resources while meeting the deadline constraints. Makespan robustness was identified as the most crucial objective for the resource allocations to fulfil. In order to quantify makespan robustness, a robustness model was applied to measure the makespan robustness of each allocation. Three forecast allocation patterns, independent consecutive forecast allocation

pattern, independent concurrent forecast allocation pattern, and independent concurrent and consecutive forecast allocation pattern, were identified. The makespan of an allocation will be affected by the forecast allocation pattern used.

Case Study 1 successfully demonstrated the effects of the forecast allocation patterns on makespan and makespan robustness, and the effectiveness and limitations of using makespan robustness as the only objective for resource allocation. The usefulness of the makespan robustness metric in supporting the resource allocation was also illustrated. The results showed that consecutive allocation pattern would increase the makespan and reduce the robustness of an allocation. Consequently, meeting the deadline and allocating all forecasts to resources within the deadline could become challenging. It is thus the least suitable forecast allocation pattern for ensembles of forecasts. Concurrent forecast allocation pattern is the most optimal pattern for ensembles of forecasts. It enables the most effective minimisation of makespan when sufficient computing resources, both in numbers and sizes, are available. Concurrent and consecutive forecast allocation pattern is the intermediate pattern in terms of makespan. It could potentially allocate all forecasts within the deadline but would not be able to minimise makespan as optimally as the concurrent forecast allocation pattern.

Case Study 1 also showed that the robustness model successfully supported the heuristic approach to quantify makespan robustness when selecting the best execution time performance resource for each forecast. The makespan robustness metric can also be used to determine if the deadline is met (the sign of the metric) and the tolerance of the allocation towards makespan perturbations (the magnitude of the metric). However, an unbalanced distribution of forecasts among the available resources occurred at times. In such cases, most forecasts were allocated only to a few resources, which could potentially increase the risk of "a single point of failure". Thus other objectives, i.e. reliability, should also be considered when allocating ensembles of forecasts to multiple resources.

A set of priority-based multi-objective heuristics for ensembles of forecasts within the urgent computing framework was thus developed. The obligations of urgent computing in supporting ensembles of forecasts, i.e. maximise the number of allocated forecasts before the deadline, and multi-objectives, i.e. makespan robustness, and site and resource reliability and energy usage in this order of priority, of the heuristics were identified. Reliability was included as an objective as a result of the observation from Case Study 1. A reliability model was thus developed to quantify resource and site reliability. As many modern HPC centres host multiple HPC resources, we identified the need to include both site and resource reliability. Since makespan is the most crucial objective, frequency scaling options that could significantly reduce makespan is included in the robustness model. Energy usage, the third objective, is used as a limiting factor to choose the frequency at which a forecast will compute on a resource. Energy usage is necessary as it represents the realistic power constraints of resource providers, and the fixed and limited computing budget dedicated to urgent computing. Thus the more energy efficient an allocation is, the more urgent computations each provider can support. Finally, an assessment model to evaluate and compare the allocation results for a given set of resources based on the defined obligations and objectives was introduced.

Case Study 2 was carried out to evaluate and compare the effectiveness of the proposed priority-based multi-objective heuristics in fulfilling the three objectives. All heuristics managed to successfully allocate all forecasts within the deadline. The assessment metrics for the two obligations were 1.0, the best possible result. The assessment metrics for makespan robustness showed that all heuristics managed to minimise makespan (maximise makespan robustness) and thus had a metric value of 1.0. Thus, all six heuristics were effective in allocating ensembles of forecasts on a given set of resources since the obligations were fulfilled and the first priority objective was maximised.

However, maximising makespan robustness does not imply that it is the best possible allocation as shown in Case Study 1 since other objectives like reliability can potentially become an issue. There is thus a need to also optimise the other two objectives, reliability and energy usage. Reliability is improved by allocating forecasts to more reliable resources and to have a balanced distribution of forecasts among the available resources and sites. Efficient energy usage is achieved by allocating forecasts to more energy efficient computing resources and when possible, use a lower clock frequency to compute a forecast. The assessment metrics for these two objectives were compared among the heuristics in Case Study 2. The results showed that reliability and energy usage came at the expense of one another. Additionally, since these two objectives have a lower priority when compared to makespan robustness, it is difficult for them to achieve the best possible assessment metric values, i.e. 1.0. However, the heuristics that included each or both of these two objectives managed to optimally increase the the assessment metric values accordingly. From the results, MM-MSRR-ME heuristic is the most optimal heuristic for fulfilling all three objectives as it managed to optimise each objective with minimum adverse influence on the other objectives. Thus, a heuristic to enable a robust, reliable and energy-aware resource allocation for ensembles of forecasts on multiple distributed heterogeneous resources is successfully provided.

In the course of carrying out this research work, a number of limitations were encountered, which needed to be addressed.

(i) Policy restrictions of resource providers affecting the support of urgent computing activities

(ii) Insufficient collaborations among urgent computing and event domain scientists

While it is possible to have technical solutions to enable urgent computing, if public resource providers cannot readily support such computations, it defeats the purpose. Consequently, it is critical for the progress of urgent computing to tackle this prevailing limitation. Section 3.5.2 thus offers a set of policy recommendations for public resource providers.

Collaborations among urgent computing and event domain scientists are critical to advance the applicability of urgent computing. The limitations faced by domain scientists, can many times be complemented by the research work, e.g. optimising the algorithms to improve makespan, of computer scientists. Thus, active collaborations among these scientists are strongly encouraged to speed up the advancement of urgent computing.

The current difficulties in finding event domain science experts to cooperate with is seemingly not new. In Trebon's urgent computing dissertation, he used a non-urgent computing astrophysics code [8, p. 36] developed in his university for his case study. It is of utmost importance to foster strong collaborations among the urgent computing researchers and event domain scientists.

In summary, the research goal to support ensembles of forecasts utilising zero hour data to produce multiple forecasts of the the same events for probabilistic forecasting before the stipulated deadlines is successfully achieved. The application of the proposed urgent computing framework for ensembles of forecasts is not limited to flash flood. It can easily be extended to other urgent events, e.g. storms and wild fires, which require ensemble forecasting.

7.2 Future Work

This dissertation has offered an urgent computing framework to realise urgent computing for ensembles of forecasts on a set of distributed heterogeneous computing resources. Since urgent computing is still a rather new field of research that requires the collaboration of multiple disciplines, there is a wide array of potential future work. Here we recommend three potential future works that can enhance and/or complement our current framework.

(i) Cost of Urgent Computations

(ii) Prediction Models

(iii) Advanced 3D Visualisation

The computation cost of a forecast typically differs from resource to resource due to the differences in hardware, personnel cost, electricity cost, etc. This cost and the cost of selecting the fully utilised computing resources, i.e. the cost of preemption to free nodes/cores [13] immediately, are not included in the resource allocation heuristics presented. One future work is to include the computation cost in the resource allocation heuristics when selecting the most appropriate resource for an ensemble of forecasts with a deterministic deadline. The cost is also useful to help determine if the urgent computations are "worthwhile", i.e. the cost of urgent computations should optimally be less than or equal to the expected savings in loss mitigation by using the urgent products for decision-making.

Prediction models [113][114][115] to evaluate the urgent products is another interesting future work to fulfil the functional requirement, data evaluation. In spite of the use of stochastic approach to compute the forecasts, there still exist inherent uncertainties that will affect the accuracy of the forecasts [116]. There is thus a need for effective prediction models to improve the interpretation of the forecasts. Prevalent techniques include the adoption of artificial neural network, regression methodology, etc. Machine learning methods [95][96] can also useful in automating the interpretation and/or to provide additional insight as to which prediction models are most effective under which set of conditions.

The prediction models are expected to be a part of the decision & coordination system of an urgent system. The collaboration with the event domain scientists will be particularly crucial when developing such models.

Advanced 3D visualisation is another interesting area for future work within the decision & coordination system to support the requirement, data evaluation. Visualising the urgent products in CAVE-like virtual environment[1] or with mobile virtual reality headsets can enable the decision makers have a swift and deep insight into the simulated datasets to make educated decision(s) effectively. Figure 7.1 illustrates a 3D visualisaton of an earthquake [15] in the CAVE-like environment. Communication with on-site civil defence services can also be potentially improved by using such techniques to share prevailing ground conditions. Initial research work [117][15][26] has began in that direction.

Figure 7.1: Advanced 3D Visualisation in CAVE-like Virtual Environment

In summary, the robust, reliable and energy-aware urgent computing framework for ensembles of forecasts can be improved by future work to include the cost of urgent computations as another objective in the heuristics, automatic prediction models and advanced 3D visualisation.

[1] https://www.lrz.de/services/v2c_en/installations_en/

Appendices

Appendix A

An Urgent Computing Visualistion Service

A production implementation of the above described ubiquitous remote visualisation service, a Virtual Network Computing (VNC) Client[1], was developed and is currently a production service at Leibniz Supercomputing centre (LRZ). This production version is developed based on the TbU approach. The service is currently offered to all LRZ local users and European users via e-Infrastructure projects, e.g. PRACE and EGI . It provides urgent users with a generic ubiquitous interface on powerful resources for evaluating urgent products for time-critical decision making. Domain scientists and civil protection services, e.g. fire-fighters, can thus access this service from their mobile devices or laptops to visualise a forecast computed on one of the HPC resources at LRZ and decide on appropriate mitigation activities or rescue plans.

A.1 Architecture

The architecture of this service is shown in Figure A.1 consisting of the ubiquitous clients, the TbU framework and the graphics clusters. The powerful graphics clusters, Graphics Cluster 1 and 2, have heterogeneous environments which can be securely accessed via this single uniform service from ubiquitous clients. The ubiquitous orchestrator enables users to have direct access to these expensive graphic clusters via any modern mobile devices, laptops and workstations with a modern browser. Each cluster has a number of graphics cards and a user will be allocated an entire graphic card while using the service.

The client front-end, which is not administered by TbU, i.e. ubiquitous orchestrator, can in principle be flexibly chosen or implemented. In this case, an in-house JavaScript de-

[1] https://rvs.lrz.de

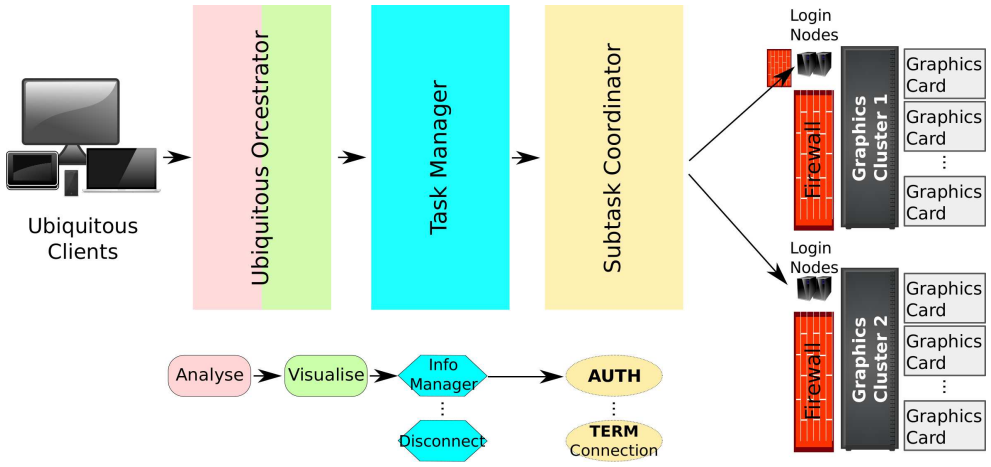
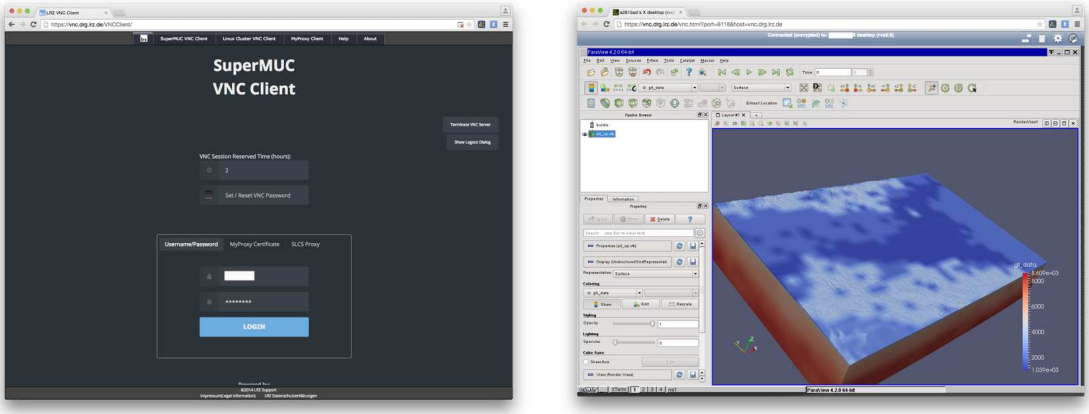


Figure A.1: Ubiquitous Visualisation Service at LRZ

velopment (Figure A.2a) for users input and an open-source VNC client software, noVNC[2], (Figure A.2b) were chosen. Figure A.2b shows the visualisation of a North Italian region where an earthquake occurred. The selection of clients should adhere to only one requirement, which is to be easily supported on ubiquitous end user devices. This particular choice for the visualisation service is thus simply a reflection of the prevailing technological trends. The client front-end is interchangeable with other alternative ubiquitous clients and should not affect the underlying TbU implementation. This is a payoff from having a three layers architecture where a separation of concerns is made possible.



(a) In-house Client Development (b) noVNC

Figure A.2: Ubiquitous Visualisation Client Interface at LRZ

[2]https://kanaka.github.io/noVNC/

A.2 Implementation with TbU Approach

To illustrate a simple generic urgent use case, three processes, compute, analyse and decide, are employed to represent an urgent workflow as shown in Figure A.3. The compute process describes the execution of a numerical simulation code for an urgent event while the analyse process shows the analysis activities of simulated result. The final process, decide, represents the act of decision making, i.e. mitigation activities, which is administered within the decision & coordination system. The visualisation service will be a part of the analyse process and correspondingly only this process is elaborated.

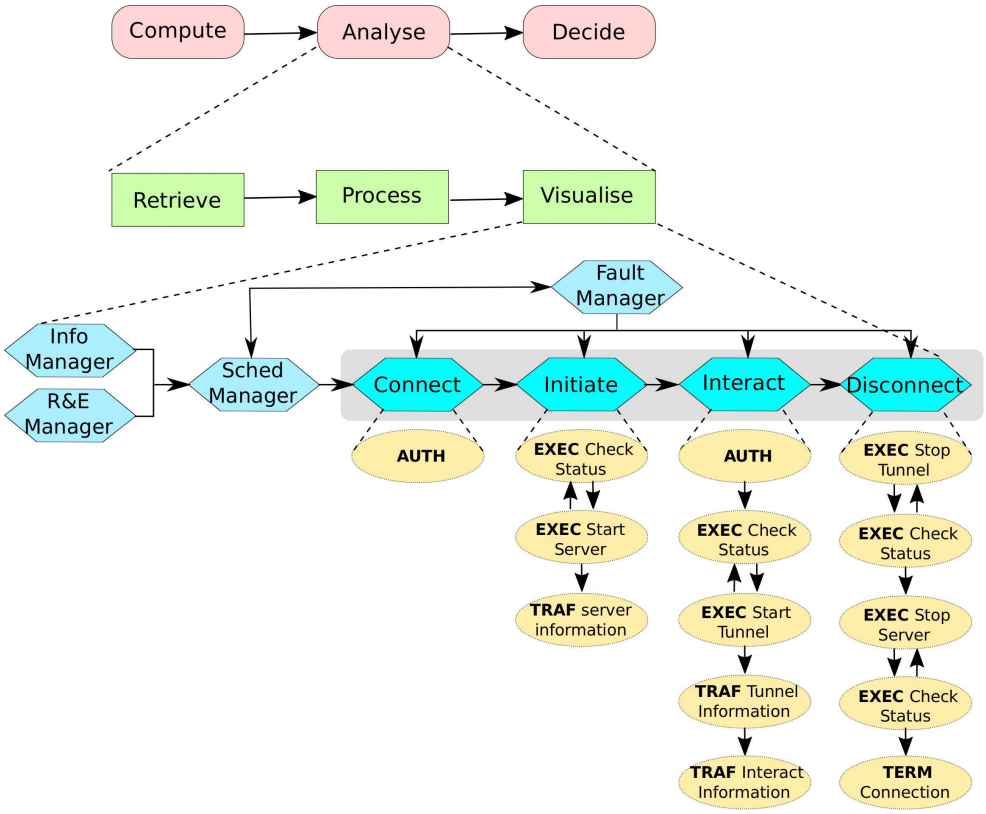


Figure A.3: TbU Analyse Process of Ubiquitous Visualistion Service

The analyse process is further broken down into three activities, retrieve, process and visualise. The retrieve activity represents the need to fetch simulated results from computing resources to the selected visualisation resource. The selection of the most appropriate resource is administered by the schedule manager. The process activity illustrates the work step to post-process simulation results to useful urgent products for visualisation. Finally the urgent products are rendered in the visualise activity. Processes and activities are administered by the ubiquitous orchestrator. Since the visualisation service will only provide the visualise functionality, only the visualise activity will be detailed. The visualise activity consists of the following core visualisation tasks, connect, initiate, interact and disconnect,

as highlighted by the gray box. These tasks should be defined by the domain scientists.

The urgent managers ensure that the urgent computing requirements are fulfilled. It is important to note that the urgent managers are integrated in a non-intrusive manner such that the core tasks can continue to function as they would when the four managers are unavailable. Each core visualisation task is described and administered with the four fundamental subtask functions. The connect task illustrates the authentication to the visualisation resource. Upon a successful connection, the initiate task starts the required visualisation service, in this case a server, on the resource. A check is performed to ensure that the required server is started. The server information is then transferred back. This information is utilised by the the interact task. Since the server is behind a firewall, the interact task will authenticate once again to start a tunnel for the interaction. A check is done to ensure the successful creation of the tunnel. The tunnel information is transferred back. Interaction information is continuously being transferred between the client and the server. When the service is no longer required, the disconnect task is activated. It will stop the tunnel and server before finally terminating the connection.

Appendix B

Preemption Approaches

Acquiring computation resources swiftly for urgent computing is crucial. Preemptive scheduling, terminating an existing job(s) to make way for an urgent job, is one of the most common approaches considered. Two cost approaches, the least cost (LC) and least disruptive (LD), are used to demonstrate the cost of preemption. The LC approach is chosen to illustrate the stand, i.e. minimum cost, of policy makers while the LD approach is selected to reflect the resource providers concerns, i.e. disrupting minimum jobs and users.

B.1 Least Cost Approach

The LC approach aims to minimise the direct cost of preempting running jobs to make way for the urgent job(s). To reduce the direct cost, the "cheapest" jobs are preempted. The "cheapness" of a job is dependent on the elapsed wallclock time and the number of nodes used.

The least cost algorithm applied is as follows.:

$$C_p(n_o) = min(CJ(n_o), LEJ(n_o)) \tag{B.1.1}$$

where

- n_o is the number of nodes that have to be preempted
- $C_p(n_o)$ is the direct cost of preempting n_o or more nodes
- $CJ(n_o)$ is the sum of the cost of preempting the cheapest running jobs to free n_o or more nodes

```
function Cheapest_Job
   input Integer  n_o
   //Jobs are sorted on cost in ascending order
   //Iterate through the sorted job list
   while n < n_o
      Add job j_i with n_i nodes to preempt_list
      n+ = n_i
   end
   if n > n_o
     KnapSack_Algo to find the most expensive
     job combination.
     Remove the job combination found from
     preempt_list(n)
     n− = n_i
   end
   if n > n_o
      KnapSack_Algo to find cheapest job
      combination within same wallclock range
      //use this set if cost is lower
   end
end
```

(a) $CJ(n_o)$

```
function Least_Elapsed_Job
   input Integer  n_o
   //Jobs are sorted on wallclock elapsed time
   in ascending order
   //Iterate through the sorted job list
   while n < n_o
      Add job j_i with n_i nodes to preempt_list
      n+ = n_i
   end
   if n > n_o
     Knapsack_Algo to find the most expensive
     job combination to remove.
     Remove the job combination found from
     preempt_list(n)
     n− = n_i
   end
end
```

(b) $LEJ(n_o)$

Figure B.1: Least Cost Algorithm

- $LEJ(n_o)$ is the sum of the cost of preempting the least wallclock elapsed jobs to free n_o or more nodes
- $min(..,..)$ selects the minimum preemption cost

The LC approach will minimise the cost of preemption by preempting the cheapest (Figure B.1a) or least elapsed wallclock time (Figure B.1b) jobs, thus favouring the preemption of smaller or newly submitted jobs respectively. If more nodes are removed than required, the list will be filtered using the Knapsack algorithm [118] where the most expensive redundant job combination will be removed from the preemption list. If more nodes are still preempted in LEJ, the Knapsack algorithm (minimisation) is additionally used to find a possible job/job combination with a lower cost within the wallclock range that is smaller and equal to the last job selected for preemption.

B.2 Least Disruptive Approach

The LD approach as shown in Figure B.2 aims to minimise the number of jobs preempted, thus the number of users affected. A job that utilises a bigger number of nodes than n_o will be preempted as compared to e.g. preempting two smaller cheaper jobs to free n_o nodes. If no jobs using $\geq n_o$ can be found, the least number of jobs to get n_o or more nodes are selected.

```
function least_disruptive
   input Integer n_o
   //Jobs ares sorted based on nodes in descending order and cost in ascending order
   Case 1: n_i == n_o
      Find job j_i with the lowest cost Where n_i == n_o
         Add job j_i to preempt_list
         n = n_i
   Case 2: n_i > n_o
      Find job j_i with the smallest job size n_i Where n_i > n_o
      And the lowest cost among jobs of the same size
         Add job j_i to preempt_list
         n = n_i
   Case 3: SUM(n_i) >= n_o
      Find jobs j_i Where SUM(n_i) > n_o
      if SUM(n_i) == n_o
         Add jobs j_i to preempt_list
         n = n_o
      else //SUM(n_i) > n_o
         //Iterate through the job list
         n = 0
         while n < n_o
            if n_i <= n_o - n
               Add job j_i to preempt_list
               n+ = n_i
            else
               Continue to iterate through the list to Find for the smallest n_i
               Where n_i > n_o - n And cost is the lowest among jobs of the same size
                  Add job j_i to preempt_list
                  n+ = n_i
            end
         end
      end
   end
end
```

Figure B.2: Least Disruptive Algorithm

Appendix C

Minimum and Maximum Values of Assessment Model

The maximum and minimum values of makespan robustness metric, site and resource reliability metrics and ETS for the given set of resources in Case Study 2 are required to compute the assessment metrics of the three objectives. In the following sections, the maximum and minimum values of the metrics and ETS values are shown.

C.1 Minimum and Maximum Values of Makespan Robustness Metric

The minimum robustness makespan metric is obtained by allocating the forecast Morrison on resource RC. This will maximise $f(\pi_j, 1) = 11326$ and correspondingly minimise the makespan robustness metric (first row of Table C.1).

The maximum robustness makespan metric is obtained by allocating the forecast Morrison on SuperMUC Phase 2 at maximum frequency 2.6 GHz. This will minimise $f(\pi_j, 1) = 10210$ (refer to equation (5.3.2)) and corresponding maximise the makespan robustness metric (second row of Table C.1).

Makespan Robustness Metric	**Value**
$\rho(\Phi, \pi_j)_{min}$	-526
$\rho(\Phi, \pi_j)_{max}$	590

Table C.1: Minimum and Maximum Values of Makespan Robustness Metric

C.2 Minimum and Maximum Values of Site and Resource Reliability Metric

The minimum (first row) and maximum site (second row) site reliability metrics are shown in Table C.2. The minimum site reliability metric is obtained by allocating all forecasts to Site2. The maximum site reliability metric is achieved by allocating three forecasts to LRZ, two forecasts to Site1 and three forecasts to Site2.

Site Reliability Metric	Value
$\rho^{site}(\phi^{reliability}, t)_{min}$	0.28
$\rho^{site}(\phi^{reliability}, t)_{max}$	0.46

Table C.2: Minimum and Maximum Values of Site Reliability Metric

The minimum (first row) and maximum site (second row) resource reliability metrics are shown in Table C.3. The minimum resource reliability metric is obtained by allocating all forecasts on one resource, SuperMUC or SuperMUC Phase 2. The maximum resource reliability metric is achieved by allocating two forecasts each to SuperMUC, SuperMUC Phase 2 and RC and one forecast each to RA and RB.

Resource Reliability Metric	Value
$\rho^{resource}(\phi^{reliability}, t)_{min}$	0.30
$\rho^{resource}(\phi^{reliability}, t)_{max}$	0.60

Table C.3: Minimum and Maximum Values of Site Reliability Metric

C.3 Minimum and Maximum Values of ETS

The minimum (first row) and maximum site (second row) ETS values are shown in Table C.3. The minimum resource reliability metric is obtained by allocating forecasts Morrison, Thompson, WSM6D, Ferrier, WSM6 and WSM3 to RC, and forecasts WMS5 and Kessler to SuperMUC Phase 2 at the minimum frequency 2.2 GHz. The ETS value is achieved by allocating all forecasts to SuperMUC at the maximum frequency 2.7 GHz.

ETS	Value
ets_{min}	89.37
ets_{max}	161.48

Table C.4: Minimum and Maximum Values of ETS

Bibliography

[1] S. A. Ackermann and J. A. Knox. *Meteoroglogy: understanding the atmosphere (4th Ed.)*. Springer Science+Business Media, 5 Wall Street, Burlington, MA 01803, USA, 2015.

[2] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications, Second Edition*. Springer, LLC, 233 Spring Street, New York, NY 10013, USA, 2011.

[3] M. O'Neill, A. R. Mikler, S. Indrakanti, C. Tiwari, and T. Jimenez. Re-plan: An extensible software architecture to facilitate disaster response planning. *IEEE transactions on systems, man, and cybernetics. Systems*, 44(12):1569–1583, 12 2014.

[4] B. Desai and A. Maskrey. *Making Development Sustainable: The Future of Disaster Risk Management. Global Assessment Report on Disaster Risk Reduction*. Geneva, Switzerland: United Nations Office for Disaster Risk Reductions (UNISDR), 2015.

[5] E. S. Epstein. Stochastic dynamic prediction. *Tellus*, 21(6):739–759, 1969.

[6] F. Atger. The skill of ensemble prediction systems. *Monthly Weather Review*, 127(9):1941–1953, 1999.

[7] S. H. Leong, A. Frank, and D. Kranzlmüller. Leveraging e-Infrastructures for Urgent Computing. In *ICCS Proceedings*, volume 18 of *Procedia Computer Science*, pages 2177–2186. Elsevier, 2013.

[8] N. Trebon. *Enabling Urgent Computing within the existing distributed computing infrastructure*. PhD thesis, University of Chicago, Aug 2011.

[9] S. H. Leong and D. Kranzlmüller. Towards a General Definition of Urgent Computing. In *ICCS Proceedings*, volume 51 of *Procedia Computer Science*, pages 2337 – 2346. Elsevier, 2015.

[10] T. L. Saaty. Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, 1(1):83–98, 2008.

[11] S. H. Leong and D. Kranzlmüller. A Task-based Ubiquitous Approach to Urgent Computing for Disaster Management. In *International Conference on Information and Communication Technologies for Disaster Management*, 2015.

[12] P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh. SPRUCE: A System for Supporting Urgent High-Performance Computing. In *Grid-Based Problem Solving Environments*, volume 239 of *Procedia Computer Science*, pages 295–311. Springer US, 2007.

[13] S. H. Leong and D. Kranzlmüller. A Case Study - Cost of Preemption for Urgent Computing on SuperMUC. In *High Performance Computing*, volume 9137 of *Lecture Notes in Computer Science*, pages 422–433. Springer International Publishing, 2015.

[14] S. H. Leong, D. Kranzlmüller, and A. Frank. A data management system to enable urgent natural disaster computing. In *EGU General Assembly Conference Abstracts*, volume 16, page 4699, May 2014.

[15] S. H. Leong and D. Kranzlmüller. Advance Visualisation and Urgent Computing. In *Intl. Symposium on Grids and Clouds 2015*, volume 17, Mar 2015.

[16] S. H. Leong, A. Parodi, and D. Kranzlmüller. A Robust Reliable Energy-Aware Urgent Computing Resource Allocation for Flash Flood Ensemble Forecasting on HPC Infrastructures for Decision Support. 2016. Submitted.

[17] S. H. Leong and D. Kranzlmüller. A Hydro-meteorological Urgent Computing Ubiquitous Framework for an Ensemble Forecast. In *CGW Workshop*, 2015.

[18] S. H. Leong and D. Kranzlmüller. Urgent Computing - A General Makespan Robustness Model for Ensembles of Forecasts. In *ICCS*, 2016. Accepted.

[19] S. H. Leong and D. Kranzlmüller. An Urgent Computing Framework for Ensembles of Forecasts on HPC Infrastructure. In *Supercomputing Frontier Conference Abstracts*, Mar 2016.

[20] M. Carpen, I. Klampanos, S. H. Leong, E. Casarotti, P. Danecek, G. Ferini, A. Gemnd, A. Krause, L. Krischer, F. Magnoni, M. Simon, A. Spinuso, L. Trani, M. Atkinson, G. Erbacci, A. Frank, H. Igel, A. Rietbrock, H. Schwichtenberg, and J.-P. Vilotte. Towards addressing cpu-intensive seismological applications in europe. In *Supercomputing*, volume 7905 of *Lecture Notes in Computer Science*, pages 55–66. Springer Berlin Heidelberg, 2013.

[21] S. Marek, S. H. Leong, K. H. Zad, L. Krischer, M. Carpene, G. Ferini, L. Trani, A. Spinuso, F. Magnoni, E. Casarotti, A. Gemünd, D. Weissenbach, I. Klampanos, and H. Igel. VERCE - CPU-intensive Applications in Seismology. In *EGU General Assembly Conference Abstracts*, volume 15, page 4483, Apr 2013.

[22] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, G. Von Laszewski, C. Lee, A. Merzky, H. Rajic, and J. Shalf. Saga: A simple api for grid applications. high-level application programming on the grid. *Computational Methods in Science and Technology*, 12(1):7–20, 2006.

[23] A. Spinuso, A. Krause, C. R. Garcia, E. Casarotti, F. Magnoni, I. A. Klampanos, L. Frobert, L. Krischer, L. Trani, M. David, S. H. Leong, and V. Muraleedharan. The VERCE Science Gateway: enabling user friendly seismic waves simulations across European HPC infrastructures. In *EGU General Assembly Conference Abstracts*, volume 16, page 6141, May 2014.

[24] A. Spinuso, A. Krause, C.R. Garcia, E. Casarotti, F. Magnoni, J. Matser, L. Krischer, L. Trani, M. David, S.H. Leong, and V. Muraleedharan. The verce science gateway: Interactive forward modeling and metadata management. In *Second European Conference on Earthquake Engineering and Seismology (2ECEES)*, Aug 2014.

[25] E. Casarotti, A. Spinuso, J. Matser, S. H. Leong, F. Magnoni, A. Krause, C. R. Garcia, V. Muraleedharan, L. Krischer, and C. Anthes. The VERCE Science Gateway: Enabling User Friendly HPC Seismic Wave Simulations. *AGU Fall Meeting Abstracts*, page A6, December 2014.

[26] S. H. Leong, C. Anthes, F. Magnoni, A. Spinuso, and E. Casarotti. Advance Visualisation of Seismic Wave Propagation and Speed Model. *Innovatives Supercomputing in Deutschland*, 13(1):34–37, 2015.

[27] M. Atkinson, M. Carpene, E. Casarotti, S. Claus, R. Filgueira, A. Frank, M. Galea, A. Gemünd, H. Igel, I. Klampanos, A. Krause, L. Krischer, S. H. Leong, F. Magnoni, J. Matser, A. Michelini, H. Schwichtenberg, A. Spinuso, and J.-P. Vilotte. VERCE delivers a productive e-Science environment for seismology research. In *11th IEEE International Conference on eScience*, Proceedings, 2015.

[28] B. Blanton, J. McGee, J. Fleming, C. Kaiser, H. Kaiser, H. Lander, R. Luettich, K. Dresback, and R. Kolar. Urgent Computing of Storm Surge for North Carolina's Coast. In *ICCS Proceedings*, volume 9 of *Procedia Computer Science*, pages 1677–1686. Elsevier, 2012.

[29] S. V. Ivanov, S. V. Kovalchuk, and A. V. Boukhanovsky. Workflow-based Collaborative Decision Support for Flood Management Systems. In *ICCS Proceedings*, volume 18 of *Procedia Computer Science*, pages 2213–2222. Elsevier, 2013.

[30] B. Balis, M. Kasztelnik, M. Bubak, T. Bartynski, T. Gubala, P. Nowakowski, and J. Broekhuijsen. The UrbanFlood Common Information Space for Early Warning Systems. In *ICCS Proceedings*, volume 4 of *Procedia Computer Science*, pages 96–105. Elsevier, 2011.

[31] G.S. Brown and D.P. Campbell. Instrument Engineering: Its Growth and Promise in Process-Control Problems. *Mechanical Engineering*, 72(2):124–127, 1950.

[32] B. Furht, D. Grostick, D. Gluch, G. Rabbat, J. Parker, and M. McRoberts. *Real-Time UNIX Systems: Design and Applications Guide*. Kluwer Academic Publishers, 1991.

[33] K.G. Shin and P. Ramanathan. Real-time computing: a new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24, Jan 1994.

[34] M. A. Cohen. A Taxonomy of Oil Spill Costs - What are the Likely Costs of the Deepwater Horizon Spill? Technical report, Resources for the Future, 2010.

[35] P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh. SPRUCE: Special PRiority and Urgent Computing Environment. Technical report, TeraGrid, 2008.

[36] P. Beckman, I. Beschastnikh, S. Nadella, and N. Trebon. *High Performance Computing and Grids in Action*, volume 16, chapter Building an Infrastructure for Urgent Computing, pages 75–95. IOS Press, 2008.

[37] V. V. Krzhizhanovskaya, N. B. Melnikova, A. M. Chirkin, S. V. Ivanov, A. V. Boukhanovsky, and P. M. A. Sloot. Distributed Simulation of City Inundation by Coupled Surface and Subsurface Porous Flow for Urban Flood Decision Support System. In *ICCS Proceedings*, volume 18 of *Procedia Computer Science*, pages 1046–1056. Elsevier, 2013.

[38] S. V. Kovalchuk and A. V. Boukhanovsky. High-Level Knowledge-Based Structures for Simulation within Urgent Computing Tasks. In *ICCS Proceedings*, volume 9 of *Procedia Computer Science*, pages 1694–1703. Elsevier, 2012.

[39] N. Palmer, R. Kemp, T. Kielmann, and H. Bal. The Case for Smartphones as an Urgent Computing Client Platform. In *ICCS Proceedings*, volume 9 of *Procedia Computer Science*, pages 1667–1676. Elsevier, 2012.

[40] A. Parodi, G. Boni, L. Ferraris, F. Siccardi, P. Pagliara, E. Trovatore, E. Foufoula-Georgiou, and D. Kranzlmueller. The "perfect storm": From across the Atlantic to the hills of Genoa. *EOS Transactions*, 93(24):225–226, Jun 2012.

[41] M. Rivi and A. Emerson. Using the IBM iDataPlex (PLX). 21st Summer School of Parallel Computing, 2012.

[42] E. Yamasaki. What We Can Learn From Japan's Early Earthquake Warning System. *Momentum*, 1:1–26, 2012.

[43] S. Marru, D. Gannon, Beckman, D. B. Weber, K. A. Brewster, and K. K. Droegemeier. LEAD Cyberinfrastructure to Track Real-Time Storms Using SPRUCE Urgent Computing. *CTWatch Quarterly*, 4(1), Mar 2008.

[44] A. Cencerrado, A. Corts, and T. Margalef. On the Way of Applying Urgent Computing Solutions to Forest Fire Propagation Prediction. In *ICCS Proceedings*, volume 9 of *Procedia Computer Science*, pages 1657–1666. Elsevier, 2012.

[45] K. K. Yoshimoto, D. J. Choi, R. L. Moore, A. Majumdar, and E. Hocks. Implementations of Urgent Computing on Production HPC Systems. In *ICCS Proceedings*, volume 9 of *Procedia Computer Science*, pages 1687–1693. Elsevier, 2012.

[46] K. V. Knyazkov, D. A. Nasonov, T. N. Tchurov, and A. V. Boukhanovsky. Interactive workflow-based infrastructure for urgent computing. 18:2223 – 2232, 2013. 2013 International Conference on Computational Science.

[47] R. Brzoza-Woch, M. Konieczny, B. Kwolek, Piotr Nawrocki, Tomasz Szydlo, and Krzysztof Zielinski. Holistic approach to urgent computing for flood decision support. 51:2387 – 2396, 2015.

[48] ERINA+. About e-Infrastructures, Nov 2012. `http://www.erinaplus.eu/index.php/about-e-infrastructures`.

[49] XSEDE. The Extreme Science and Engineering Discovery Environment 2011-2012 Annual Highlights. Technical report, XSEDE, 2012.

[50] A. Musa, H. Matsuoka, O. Watanabe, Y. Murashima, S. Koshimura, R. Hino, Y. Ohta, and H. Kobayashi. A Real-Time Tsunami Inundation Forecast System for Tsunami Disaster Prevention and Mitigation. In *The International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2015.

[51] H. Kobayashi. One-Year Experience with SX-ACE. 22nd Workshop on Sustained Simulation Performance, Dec 2015.

[52] S. Koshimura, T. Oie, H. Yanagisawa, and F. Imamura. Developing Fragility Functions for Tsunami Damage Estimation using Numerical Model and Post-Tsunami data from Banda Aceh, Indonesia. *Coastal Engineering Journal*, 51(03):243–273, 2009.

[53] A. Geist and C. Engelmann. Development of Naturally Fault Tolerant Algorithms. Technical report, Oak Ridge National Laboratory, 2002.

[54] M. Hegland. Multidimensional problems and fault tolerance. International Symposium on Parallel and Distributed Computing (ISPDC 2012), 2012. Invited talk.

[55] T. Aboulnasr and Q. Pan. Data-dependent partial update adaptive algorithms for linear and nonlinear systems. In *Signal Processing Conference, 2005 13th European*, pages 1–4, Sept 2005.

[56] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM.

[57] S. V. Kovalchuk, P. A. Smirnov, S. V. Maryin, T. N. Tchurov, and V. A. Karbovskiy. Deadline-driven Resource Management within Urgent Computing Cyberinfrastructure. 18:2203–2212, 2013.

[58] K. Kurowski, A. Oleksiak, w. Piatek, and J. Weglarz. Impact of Urgent Computing on Resource Management Policies, Schedules and Resources Utilization. In *ICCS Proceedings*, volume 9 of *Procedia Computer Science*, pages 1713–1722. Elsevier, 2012.

[59] D. Nasonov and N. Butakov. Hybrid Scheduling Algorithm in Early Warning Systems. *Procedia Computer Science*, 29:1677 – 1687, 2014. 2014 International Conference on Computational Science.

[60] O. H. Ibarra and C. E. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *J. ACM*, 24(2):280–289, Apr 1977.

[61] L. Sha, T. Abdelzaher, K. E. Arzn, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems*, 28(2-3):101–155, 2004.

[62] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Design Automation Conference, 1999. Proceedings. 36th*, pages 134–139, 1999.

[63] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Design Automation Conference, 2002. Proceedings. 39th*, pages 183–188, 2002.

[64] R. Ge, X. Feng, and K. W. Cameron. Performance-constrained distributed dvs scheduling for scientific applications on power-aware clusters. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 34–34, Nov 2005.

[65] A. M. Methta, J. Smith, H. J. Siegel, A. A. Maciejewski, A. Jayaseelan, and B. Ye. Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness. *The Journal of Supercomputing*, 42(1):33–58, 2007.

[66] J. Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3):034008, 2008.

[67] J. Koomey. Growth in Data center electricity use 2005 to 2010. Technical report, Stanford University, 2011.

[68] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor & Study Lead, 2008.

[69] H. Shoukourian. *Adviser for Energy Consumption Management: Green Energy Conservation* . PhD thesis, Technische Universität München, Jul 2015.

[70] H.F. Sheikh, H. Tan, I. Ahmad, S. Ranka, and P. Bv. Energy- and performance-aware scheduling of tasks on parallel and distributed systems. *J. Emerg. Technol. Comput. Syst.*, 8(4):32:1–32:37, November 2012.

[71] E. Seo, J. Jeong, S. Park, and J. Lee. Energy efficient scheduling of real-time tasks on multicore processors. *Parallel and Distributed Systems, IEEE Transactions on*, 19(11):1540–1552, Nov 2008.

[72] S. Srinivasan and N. K. Jha. Safety and reliability driven task allocation in distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, 10(3):238–251, 1999.

[73] X. Qin and H. Jiang. A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. *Journal of Parallel and Distributed Computing*, 65(8):885 – 900, 2005.

[74] Andrea Rossa, Katharina Liechti, Massimiliano Zappa, Michael Bruen, Urs Germann, Gnther Haase, Christian Keil, and Peter Krahe. The {COST} 731 action: A review on uncertainty propagation in advanced hydro-meteorological forecast systems. *Atmospheric Research*, 100(23):150 – 167, 2011. Uncertainty Propagation in Advanced Hydro-Meteorological Forecast Systems.

[75] L. Ferraris, R. Rudari, and F. Siccardi. The uncertainty in the prediction of flash floods in the northern mediterranean environment. *Journal of Hydrometeorology*, 3(6):714–727, 2016/01/24 2002.

[76] J. C. Bartholmes, J .Thielen, M. H. Ramos, and S. Gentilini. The European Flood Alert System EFAS Part 2: Statistical skill assessment of probabilistic and deterministic operational forecasts. *Hydrology and Earth System Sciences*, 13(2):141–153, 2009.

[77] L. Alfieri, P. Burek, E. Dutra, B. Krzeminski, D. Muraro, J. Thielen, and F. Pappenberger. GloFAS — Global ensemble streamflow forecasting and flood early warning. *Hydrology and Earth System Sciences*, 17(3):1161–1175, 2013.

[78] European Centre for Medium-Range Weather Forecasts (ECMWF). Forecasts, Dec 2015. `http://www.ecmwf.int/en/forecasts`.

[79] Deutscher Wetterdienst (DWD). Ensemble Prediction, Dec 2015. http://www.dwd.de/EN/research/weatherforecasting/num_modelling/04_ensemble_methods/ensemble_prediction/ensemble_prediction_node.html.

[80] National Centers for Environmental Prediction (NCEP). Global Ensemble Forecast System (GEFS), Dec 2015. https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/global-ensemble-forecast-system-gefs.

[81] China Meteorological Administration (CMA). An introduction of ensemble forecast of CMA, May 2015. http://www.cma.gov.cn/en2014/news/Features/201505/t20150505_281493.html.

[82] Bureau of Meteorology. Climate Model Details, Dec 2015. http://www.bom.gov.au/climate/model-summary/model-summary-table.shtml.

[83] Japan Meteorological Agency (JMA). JMA's Ensemble Prediction System (Products of GPC Tokyo), Dec 2015. http://www.bom.gov.au/climate/model-summary/model-summary-table.shtml.

[84] L. Alfieri, L. Feyen, F. Dottori, and A. Bianchi. Ensemble flood risk assessment in europe under high end climate scenarios. *Global Environmental Change*, 35:199 – 212, 2015.

[85] B. Revilla-Romero, J. Thielen, P. Salamon, T. De Groeve, and G. R. Brakenridge. Evaluation of the satellite-based Global Flood Detection System for measuring river discharge: influence of local factors. *Hydrology and Earth System Sciences*, 18(11):4467–4484, 2014.

[86] A. Pieri, J. von Hardenberg, A. Parodi and A. Provenzale. Sensitivity of Precipitation Statistics to Resolution, Microphysics, and Convective Parameterization: A Case Study with the High-Resolution WRF Climate Model over Europe. *J. Hydrometeor*, 16(4):1857–1872, 2015/08/18 2015.

[87] C. A. Doswell, H. E. Brooks, and R. A. Maddox. Flash Flood Forecasting: An Ingredients-Based Methodology. *Weather and Forecasting*, 11(4):560–581, 2015/10/30 1996.

[88] A. Hally, O. Caumont, L. Garrote, E. Richard, A. Weerts, F. Delogu, E. Fiori, N. Rebora, A. Parodi, A. Mihalović, M. Ivković, L. Dekić, W. van Verseveld, O. Nuissier, V. Ducrocq, D. D'Agostino, A. Galizia, E. Danovaro, and A. Clematis. Hydrometeorological multi-model ensemble simulations of the 4 november 2011 flash flood event in genoa, italy, in the framework of the drihm project. *Natural Hazards and Earth System Sciences*, 15(3):537–555, 2015.

[89] E. Fiori, A. Comellas, L. Molini, N. Rebora, F. Siccardi, D. J. Gochis, S. Tanelli, and A. Parodi. Analysis and hindcast simulations of an extreme rainfall event in the mediterranean area: The genoa 2011 case. *Atmospheric Research*, 138:13 – 29, 2014.

[90] S. Davolio, F. Silvestro, and P. Malguzzi. Effects of increasing horizontal resolution in a convection-permitting model on flood forecasting: The 2011 dramatic events in liguria, italy. *Journal of Hydrometeorology*, 16(4):1843–1856, 2016/01/24 2015.

[91] S. Anquetin, J.-D. Creutin, G. Delrieu, V. Ducrocq, E. Gaume, and I. Ruin. Increasing the forecasting lead-time of weather driven flash-floods. Technical report, April 2004. Rapport H01/812/02/D9056/AG/ct de 47 p. [Annexe : p. 32 à 47].

[92] B. Paplinska-Swerpel, L. Paszke, W. Sulisz, and R. Bolanos. Application of statistical methods for the prediction of extreme wave events. *Journal of Hydraulic Research*, 46(sup2):314–323, 2008.

[93] G. W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1–3, 2015/11/18 1950.

[94] W. Marzocchi, J. D. Zechar, and T. H. Jordan. Bayesian Forecast Evaluation and Ensemble Earthquake Forecasting. *The Bulletin of the Seismological Society of America*, 102:2574–2584, December 2012.

[95] K. Rasouli, W. W. Hsieh, and A. J. Cannon. Daily streamflow forecasting by machine learning methods with weather and climate inputs. *Journal of Hydrology*, 414–415:284 – 293, 2012.

[96] C. Marzban and G. J. Stumpf. A neural network for tornado prediction based on doppler radar-derived attributes. *Journal of Applied Meteorology*, 35(5):617–626, 2015/11/18 1996.

[97] National Science Foundation. The TeraGrid Community Steps Up to Help Japan in Crisis, Nov 2012. `https://www.nsf.gov/discoveries/disc_summ.jsp?cntn_id=119412&org=NSF%20Oil%20spill%20simulation`.

[98] J. Clark. Amazon Web Services: The hidden bugs that made AWS outage worse, Jul 2012. `http://www.zdnet.com/amazon-web-services-the-hidden-bugs-that-made-aws-outage-worse-7000000186/`.

[99] S. Gennies, A. Funk, M. Schlegel, and D. Dehmer. Hochwasser-Bilanz 2013 Wie schlimm war die Flut wirklich?, Jun 2013. `http://www.tagesspiegel.de/politik/hochwasser-bilanz-2013-wie-schlimm-war-die-flut-wirklich/8416770.html`.

[100] K. Ujikane. Japan Forecasts Earthquake Damage May Swell to $309 Billion, Mar 2011. `http://www.bloomberg.com/news/articles/2011-03-23/japan-sees-quake-damage-bill-of-up-to-309-billion-almost-four-katrinas`.

[101] D.K. Nanto, W.H. Cooper, and J.M. Donnelly. Japans 2011 Earthquake and Tsunami- Economic Effects and Implications for the United States. Technical report, Congressional Research Service, 2011.

[102] M. L. Dolfman, S. F. Wasser, and B. Bergman. The effects of Hurricane Katrina on the New Orleans economy. Technical report, Bureau of Labor Statistics of the U.S. Department of Labor, 2007.

[103] E. Gica, M. C. Spillane, and V. V. Titov. Development of the forecast propagation database for NOAA's short-term inundation forecast for tsunamis (SIFT). Technical Report OAR PMEL-139, U.S. Dept. of Commerce, National Oceanic and Atmospheric Administration, Office of Oceanic and Atmospheric Research, Pacific Marine Environmental Laboratory, 2008.

[104] R. E. Litan, J. W. Jacobs, W. D. Iwan, and S. D. Parker. The Impacts of Natural Disasters: A Framework for Loss Estimation. Technical report, Committee on Assessing the Costs of Natural Disasters - Board on Natural Disasters and Commission on Geosciences Environment and Resources - National Research Council, 1999.

[105] R. C. Tausworthe. The work breakdown structure in software project management. *Journal of Systems and Software*, 1:181–186, 1979.

[106] IEEE Standard for Developing Software Life Cycle Processes. *IEEE Std 1074-1991*, pages 1–, 1992.

[107] IEEE Standard for Developing Software Life Cycle Processes. *IEEE Std 1074-1997*, pages i–, 1998.

[108] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim. Measuring the robustness of a resource allocation. *Parallel and Distributed Systems, IEEE Transactions on*, 15(7):630–641, Jul 2004.

[109] G. Strang and K. Borre. *Linear Algebra, Geodesy, and GPS*. Wellesley-Cambridge, 1997.

[110] A. Auweter, A. Bode, M. Brehm, L. Brochard, N. Hammer, H. Huber, R. Panda, f. Thomas, and T. Wilde. A case study of energy aware scheduling on supermuc. In *Supercomputing*, volume 8488 of *Lecture Notes in Computer Science*, pages 394–409. Springer International Publishing, 2014.

[111] D. Usman I.B. Mohamad. Standardization and Its Effects on K-Means Clustering Algorithm. *Research Journal of Applied Sciences, Engineering and Technology*, 6(17):3299–3303, 2013.

[112] J. Dudhia. 11th WRF Users' Workshop - Microphysics Options in WRF. 2010.

[113] S. A. Cheong, T. L. Tan, C.-C. Chen, W.-L. Chang, Z. Liu, L. Y. Chew, P. M. A. Sloot., and N. F. Johnson. Short-term forecasting of taiwanese earthquakes using a universal model of fusion-fission processes. *Scientific Reports*, 4:3624 EP –, 01 2014.

[114] E. E. Ebert, U. Damrath, W. Wergen, and M. E. Baldwin. The wgne assessment of short-term quantitative precipitation forecasts. *Bulletin of the American Meteorological Society*, 84(4):481–492, 2016/01/21 2003.

[115] V. Viswanathanand C. E Lee, M. H. Lees, S. A. Cheong, and P. M. A. Sloot. Quantitative comparison between crowd models for evacuation planning and evaluation. *European Physical Journal B*, 87:27, Feb 2014.

[116] E. Toth, A. Brath, and A. Montanari. Comparison of short-term rainfall prediction models for real-time flood forecasting. *Journal of Hydrology*, 239(14):132 – 147, 2000.

[117] E. Pajorová, L. Hluchý, and C. Anthes. 3D Geovisualization Service for Grid-oriented applications of Natural Disasters. In *16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision - Poster Proceedings (WSCG '08)*, pages 1–4, Plzen, Czech Republic, Feb 2008.

[118] M. A. Trick. A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. In *Annals of Operations Research*, pages 113–124, 2001.

Acknowledgement

I would like to thank my parents, Leong Fook Choi and Kee Hong Eng, and two siblings, Leong Siew Yin and Leong Guotang, the driving forces behind the completion of this dissertation, for their persistent support and encouragement (including the teasing of course).

Next, I would like to thank Prof. Dr. Dieter Kranzlmüller, for his constant guidance, support and not forgetting patience while supervising this work. His involvement is imperative to the success of this work.

I would also like to thank Prof. Dr. Hans-Joachim Bungartz for providing guidance and support. His advices are invaluable not only for this dissertation but beyond to enable sustainability for the research activities in this area of research.

There are many others, from within Leibniz Supercomputing Centre (LRZ) and outside, who had taken valuable time to discuss my research work with me, provided guidance, valuable inputs, opportunities, etc. that help complete this dissertation.

From LRZ, I would like to thank Christoph Anthes, Reinhold Bader, Hayk Shoukourian, Helmut Heller, Anton Frank, Matthias Brehm and Herbert Huber. I would also like to thank the group members of VER, in particular Jens Weismüller for translating the abstract to German, and Matteo Lanati and Shaila Roessle-Blank, for their constant support.

Last but not least, I would like to thank Antonio Parodi for discussing hydrometeorology problems and providing the case study problem, Ionel Muntean, for guidance and encouragement to kick start my first publication, Nick Trebon, for answering my enquiries and giving me words of encouragement, Emanuele Casarotti for discussing seismological problems with me, Michael Johnson for proofreading, Mark Yampolskiy and Ilya Saverchenko, for their mental support.

Thank you everyone!

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Leong, Siew Hoon

Name, Vorname

Garching bei München

Ort, Datum

Unterschrift Doktorand/in

Formular 3.2