

Managing the Forecast Factory

Laura Bright

David Maier

Bill Howe

Department of Computer Science

Portland State University

Portland, Oregon

{bright,maier,howe}@cs.pdx.edu

Abstract

The CORIE forecast factory consists of a set of data product generation runs that are executed daily on dedicated local resources. The goal is to maximize productivity and resource utilization while still ensuring timely completion of all forecasts. Many existing workflow management systems address low-level workflow specification and execution challenges, but do not directly address the high-level challenges posed by large-scale data product factories. In this paper we discuss several specific challenges to managing the CORIE forecast factory including planning and scheduling, improving data flow, and analyzing log data, and point out their analogs in the “physical” manufacturing world. We present solutions we have implemented to address these challenges, and present experimental results that show the benefits of these solutions.

1 Introduction

Scientific workflows have become an increasingly important part of research in domains as diverse as biology, astronomy, physics, and environmental modeling. Characteristics and requirements of workflow systems in these domains can vary widely. Some workflow management systems emphasize ease of specification and use or interactive processing, and timely workflow completion is desirable but not critical. Other workflow management systems repeatedly perform the same large-scale computations subject to time constraints. We refer to the latter as a data product factory setting, and we focus on planning and production management for daily execution of thousands of workflow instances for numerical simulations and derived products.

The CORIE environmental observation and forecasting system models the physical properties of multiple rivers and associated bodies of water throughout the coastal United States. An important component of the CORIE system is what we term “the forecast factory”, which runs daily fore-

casts to predict near-term conditions of these bodies of water, including temperature, salinity, and velocity. Each particular forecast simulates a two-day period, and involves tasks to generate tens to hundreds of instances of dozens of types of derived data products. While one might conceive of the aggregate collection of tools associated with one forecast as a single, massive workflow, we tend to think of each data product being generated by a separate workflow instance. In the sequel, we will refer to the simulation execution and associated product workflows as a “product run” or simply a “run”. Keeping all runs executing smoothly is critical to ensure that all needed data products are delivered on time. While there is a large body of research in scientific workflows that addresses challenges to specifying and executing individual workflows, managing the entire collection of runs in the forecast factory presents new challenges.

To provide insight into the class of management activities we address, we provide a brief overview of Manufacturing Resource Planning (MRP) in real-world factories, and discuss analogies to the forecast factory. Figure 1 illustrates the hierarchy of activities involved with MRP, adapted from an industrial engineering textbook [9]. The graph distinguishes between long-range, intermediate-range, and short-range planning activities, which we describe below.

Long-range planning (Figure 1(a)) operates at the level of product *families* and seeks to answer two questions: “Which products are customers interested in?” and “Which products are we capable of making?”. Each geographical region forecasted by the CORIE factory is analogous to a product family. Each forecast manufactures thousands of individual products; the term *Aggregate planning* refers to this level of abstraction. Adding new forecasts to the production schedule must consider long-term allocation of staff and compute resource planning. We expect that our techniques will come to steer long-range planning activities, though our current focus is on intermediate-range planning.

Intermediate-range planning (Figure 1(b)) corresponds to implementation of the long-range strategy. Tasks at this level include capacity planning and production scheduling

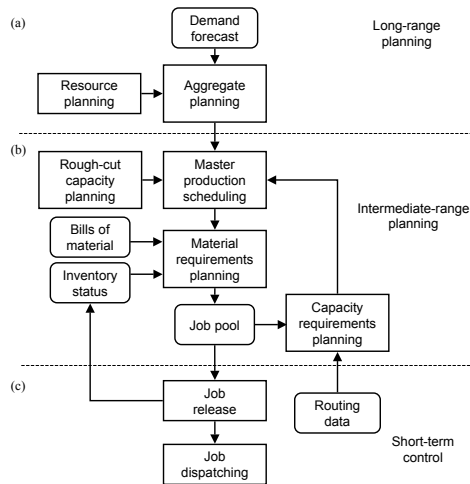


Figure 1. Manufacturing Resource Planning (MRP) Hierarchy

(described below), as well as tasks related to design and layout of the factory. *Rough-cut capacity planning* (RCCP) determines the feasibility of the long-term aggregate plan by considering demand and capacity estimates by period. A *master production schedule* is derived from the aggregate planning and the RCCP. The corresponding activity in the CORIE forecast factory is to estimate the running time (makespan) of all forecasts for a day and compare it to available computing capacity, to ensure the collective resource requirements do not exceed the total capacity. Using past statistics for similar tasks, the forecast factory derives a *bill of resources*: an estimate of the time required for each step to complete. In addition to the bill of resources, a traditional *bill of materials* (BOM) is considered during capacity planning. The BOM is a tree of subassemblies used to build an end item. Specification and maintenance of the BOM is the responsibility of workflow specification tools in our scientific workflow analog. Individual workflow specification is an important problem and the subject of many research efforts [1, 12], but is not our focus.

Finally, short-term control (Figure 1(c)) includes job scheduling tasks at the level of individual workflows. Many existing workflow execution systems [5, 7, 15] provide this functionality. Most scientific workflow management systems we have encountered tend to involve specification and scheduling of individual workflows, both of which we consider part of short-term control rather than intermediate-range or long-range planning. Instead, our research addresses management of large-volume, long running production campaigns, with guidance from our factory metaphor.

There is a large body of research in resource manage-

ment and workflow scheduling in cluster and Grid environments. These efforts are largely complementary to ours. First, Grid and cluster management of scientific workflows is typically decentralized, and resources are shared by a diverse group of users and workflows with varying resource and QoS requirements. Distributed management contrasts with the centralized management and planning in a factory. Second, Grid and cluster computing solutions typically allow users to specify resource requirements for a job, and a scheduler locates and secures the needed resources and schedules the job for execution. In a data product factory, however, the available resources are known ahead of time, and the set of product runs must be tailored to complete on these resources within the allotted time frame.

We discuss several specific intermediate-range planning challenges analogous to factory management that we address in our forecast factory:

Plant Layout. A fundamental task in Facilities Planning is to find a layout of work cells such that two competing goals are balanced: reduce or eliminate unnecessary movement of material but maintain utilization when material must be moved from an overloaded work cell to an idle one. Similarly, we wish to minimize unnecessary data transfer in the forecast factory while maintaining high machine utilization. For example, in some cases we want to execute a series of tasks that access the same data on the same node, to minimize the cost of copying large files across the network. However, in other cases we find that a pipelined execution model that moves data from a primary node to a secondary node can avoid a bottleneck on the primary node.

Capacity Planning and Scheduling. In Figure 1, jobs produced by the Material Requirements Planning process are subject to *capacity requirements planning* (CRP). CRP is a more detailed process than the optimistic RCCP. It schedules products to individual workcells, taking into account factors such as materials arrival and work in progress (WIP), and predicting job completion times. The corresponding activity in the forecast factory is assigning runs to particular compute nodes, taking into account availability of inputs such as atmospheric predictions and runs still executing from the previous day (WIP), while meeting desired finish times.

Statistical Process Control. Plant Managers use statistical process control to reduce uncertainty on the factory floor. For example, process time variability, regardless of source, results in increased work-in-progress. An analog in the forecast factory is that changes to forecast code versions, nodes, and parameters may change forecast running times and make the existing node assignments suboptimal. To prevent problems before they start, historical data can be used as a baseline to help determine possible effects of changes. In the forecast factory, we harvest a variety of measurements from log files and use the data to model run

times. A graphical display of these data aids interpretation.

While the factory metaphor for forecast generation helps elucidate many of the management activities involved, it does not capture all the nuances. In particular, the “goods” being manufactured in the forecast factory are “perishable” or “time-sensitive” in the sense that they are much less valuable if they arrive after the time period they forecast (though there are still some uses for them). This aspect of forecast generation is perhaps better understood by comparison to a newspaper producing various regional editions. There is a tension between including the latest information – election results, financial prices, sports scores – and getting an edition out before forward-looking information gets stale: movie listings, TV schedules, sales ads. In the forecast domain, there is a tension between getting the latest inputs to the process – river flows, atmospheric forcings – and having the forecast available before the time period it predicts.

Further, given a certain level of staffing and a given printing capacity, there are limits on how many distinct editions a newspaper can produce. Having idle capacity at mid-morning doesn’t mean the newspaper can necessarily add another edition and have it be timely for the current day. In the forecast domain, forecasts and their derivative products generally need to be available for the current day when people local to the forecast region need them, which can be quite early in the morning for, say, a fishing-boat captain.

There is one final aspect of forecast production that the newspaper parallel (at least for conventional hard-copy newspapers) doesn’t really reflect: It can be useful to deliver forecasts and their derived products incrementally. Even though a forecast for the current day might not finish until, say, 10am in the morning, the portion of the forecast completed by 7am might cover the time period up until noon. Thus, forecast users are interested in the completed part of the forecast for current and near-term conditions. Hence, it is normal to move forecasts and products incrementally from the compute nodes to the servers where they are publicly available. It would be as if your newspaper was delivered a couple of pages or sections at a time – first international news and yesterday’s stock quotes, then the national news, then local news and late sports scores, and so forth.

In this paper we present high-level factory solutions we have developed to manage the CORIE forecast factory. We believe many of our resource management, scheduling, and data flow solutions will be applicable to other data product factories. Our solutions are independent of any particular implementation, and can be used in conjunction with existing scientific workflow tools.

This paper is organized as follows: Section 2 gives an overview of the CORIE system with an emphasis on the CORIE forecast factory and the unique challenges it poses. We survey related work in Section 3, and present the solu-

tions we have implemented to address the challenges of the forecast factory in Section 4. We conclude in Section 5.

2 CORIE Overview

CORIE [2] is an environmental observation and forecasting system that models many coastal regions throughout the United States. The system includes daily forecasts to predict near term conditions as well as calibration runs and hindcasts that are run retroactively for a fixed period of time. Eventually the system will include all rivers on the west, east and gulf coasts of the U.S., which will comprise about 50-100 forecasts. We focus on forecasts due to the challenges they pose to factory management.

Forecast simulations model regions for a fixed time period, typically two days, and models conditions at specific *timesteps*, e.g., once every 30 seconds. Shorter timesteps provide more detailed information but increase the total number of steps that must be run, thus increasing the running time of the forecast. A *mesh* defines the points in the region to be modeled, and can be specified at any granularity. Finer-grained meshes provide a more detailed model but may increase the forecast running time. Forecast runs also incorporate some real-time observation data, thus their starting times may be constrained.

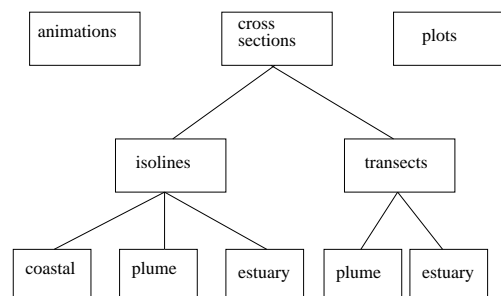


Figure 2. Partial classification of data products in the CORIE forecast factory

Simulation runs for different regions range in running time from just a few hours to over a day. These runs produce multiple large output files that contain data on various features, such as salinity, temperature, and velocity. As the simulation run progresses, data is appended to output files. Data products are generated incrementally from one or more of these files. Figure 2 presents a classification of some of the data products in a forecast run. There are many specific products in each class, such as for varying water depths and variables, and a different set of products from one or more of these classes is generated for each forecast.

2.1 Challenges

Several characteristics of the CORIE forecast workflow management system present unique challenges. First, the number of runs and data products are limited currently by the available resources. When additional resources become available, additional forecasts can be run. Alternatively, existing forecasts can be run at a finer timestep granularity or on a finer mesh, or additional data products can be added to a product run. Similarly, if existing resources are insufficient to complete all forecasts in a timely manner, some forecasts may need to be run at a coarser timestep granularity or the number of derived data products may be reduced.

Another important characteristic is that timely completion of workflows is critical. Since these workflows are forecasts, they have the most value when they complete well before the time period they are forecasting, and the value of their data products declines rapidly as the time of the forecast period approaches. In addition, compute node assignment and scheduling decisions must consider the effect of the decision on all forecast runs, since early completion of one run may delay the completion of another. This centralized management approach differs from many distributed systems that execute jobs from multiple users who are competing for available resources.

Finally, the set of forecast runs is dynamic in nature. Forecasts are continually being added and modified. Several simulation models are in current use, and developers are continually refining the model and deploying new code versions. Changes to timestep granularity, meshes, code versions, and data products can all affect a forecast's running time. Changes to a forecast may require it to be reassigned to a different node to ensure its timely completion as well as those of other forecasts. In addition, if a node becomes temporarily unavailable, forecasts scheduled to run on it must be reassigned and executed as early as possible. To accommodate the displaced forecasts, other runs may need to be reassigned as well.

2.2 Existing Implementation

Currently the forecast factory is managed manually by a team of programmers and computational scientists. Forecasts are allocated to specific nodes using scripts that stage in all needed input files, launch the workflow, and stage out data products to a shared repository. When a forecast is added, programmers may manually shift existing forecasts to different nodes to optimize forecast overall completion time. Due to the difficulty of estimating forecast running times at different nodes, this process may be repeated for several days until a good mapping is found. Currently there are 6 dedicated forecast nodes, each with two CPUs, and 10 forecast runs, and new nodes will be added as the number

of forecasts grows to the expected 50-100 per day.

Currently the simulation model and data product generation for a single forecast run concurrently at the same node. Data products are incrementally computed as additional model data is appended to output files, so initial data products are available before an entire run has completed. While this architecture minimizes data movement, data product generation consumes CPU cycles and memory and may interfere with model execution, thus delaying forecast completion times. Thus, alternate data flow architectures may better leverage available resources and reduce the completion time of runs. We note that in the current factory implementation, there is generally little benefit to generating data products for a single forecast concurrently at multiple nodes, due to high data transfer overhead and limited node availability. In the future, however, parallel code versions or increased node capacity may make partitioning different data products across multiple nodes a more attractive option, so we plan to revisit this issue.

As noted above, code versions of the simulation models, timestep granularities, and meshes are continually being modified. Data from past forecast executions can aid programmers in estimating the effects of such changes on forecast execution times and improve the ability of programmers to determine a good mapping of forecasts to nodes. However, most of this historical data is currently stored in log files. Storing data in files makes it difficult to extract the relevant information or observe long-term trends.

Based on the above challenges, we had several specific goals to improve management of the forecast factory:

- **Managing Node Assignments:** Managing node assignments is important for capacity planning and scheduling. An interface to help users assign forecast runs to nodes can improve the ease of adding new forecasts to the system. In addition, heuristics to automate assignments of runs to nodes can improve fault tolerance.
- **Improving Data Flow:** Developing architectures that can generate data products quickly and better utilize available memory and CPU cycles can significantly improve the efficiency of the forecast factory. Also, incremental delivery of data products can make initial results available sooner, even if the entire workflow has not yet completed.
- **Leveraging Historical Data:** Analyzing log data of past forecast runs can provide important insights into how changes to forecasts, e.g., code versions, node assignments, timestep granularities, will affect forecast running times. Thus, one of our goals is to develop an interface to allow users to efficiently analyze this data.

We emphasize that while this paper refers to CORIE forecast runs at a high level of abstraction, i.e., a simulation run followed by data product generation, the underlying data product specification and generation process has a high degree of complexity. A different set of data products is generated for each forecast, and specification of these products is currently done manually. Further, data products are typically computed incrementally as simulation output is generated, so generating a single product requires a large number of invocations of a single program. In addition, many data products operate on multiple simulation outputs simultaneously, and there may be dependencies among data products. Many existing workflow specification tools [1, 12] may be useful to specify the computation of these data products, and indeed we expect that such tools could play an important role in automating data product generation. However, these lower-level functions are not the focus of this paper.

3 Related Work

Much of the existing research in scientific workflow execution in clusters and Grids focuses on fair sharing of resources among competing users or task scheduling at the level of individual workflows. However, there has been little research in coordinating a set of workflows. Yu and Buyya [17] present a taxonomy of scientific workflow systems on the Grid and classify many existing systems. They classify workflow management systems along several dimensions including centralized vs. decentralized scheduling architecture, global vs. local decision making, and performance-driven vs. market driven strategies. While our forecast factory can be classified broadly as centralized scheduling, global decision making, and performance-driven strategy, it differs from other systems in these categories [5, 7] in that it considers these challenges at the level of multiple runs.

Pegasus [7] maps complex workflows onto a Grid while adapting to changes in resource availability. GrADS [5] aims to minimize the completion time of a set of workflow tasks, and can reschedule workflow tasks as needed. Nimrod-G [3] is a market-driven solution to scheduling scientific workflow tasks. Workflow completion times are determined by how much a user is willing to pay, and users can pay more to guarantee completion by a deadline. While this system (and other market-driven systems) shares our goal of completing workflows by user-specified deadlines, it is designed for sharing resources among competing users rather than centralized workflow management.

Many cluster computing tools, e.g., Torque/Maui [15] provide facilities for advance reservations to secure compute resources for a workflow prior to execution. Condor [10] aims to exploit idle workstations and is well-suited for long-running on-demand background tasks. While such

tools are useful to ensure resource availability and fair allocation of shared resources, they do not ensure completion of workflows prior to a deadline, and do not determine what data products can be generated using available resources.

Chimera [8] introduces the concept of *virtual data*, which allows data products to be defined before they are actually needed, and materialized on demand. While Chimera also supports batch workflow execution, it relies on underlying Grid mechanisms to reserve resources and manage execution. Workflow management in GriPhyN [6] addresses many of the same challenges we do, including resource management, data staging, and planning, but is geared towards globally distributed users.

Many tools, including Kepler [1] and Taverna [12], focus on the specification of scientific workflows. In contrast, in this paper we focus on managing the execution of multiple workflows. GridDB [11] provides facilities for workflow specification and processing but does not explicitly assign workflow tasks to nodes, instead relying on existing middleware [10].

The Sloan Digital Sky Survey [14] is a factory that processes large volumes of astronomical data. While not explicitly deadline constrained, efficient pipelining of data is important and the system aims to leverage all available resources to generate the needed data products.

Many tools, including GrADS [5] and ASKALON [16], incorporate data location and data transfer overhead into workflow scheduling decisions. These tools share our goal of improving data flow in scientific workflow execution. However, they may not explicitly consider alternative architectures as we do in Section 4.2.

Shankar et al. [13] advocate augmenting databases to manage scientific workflows. The authors suggest that database technology could improve workflow planning, scheduling, and cluster management. However, incorporating database functionality into legacy workflow code may have high overhead. As in our work, the authors suggest that querying data from log files such as the status of machines and jobs in a cluster could improve workflow monitoring and execution. We extend this idea in Section 4.3.2 and use data from past runs to manage a set of workflows.

4 Solutions

We now present several solutions we have implemented to address challenges posed by the forecast factory.

4.1 Node Assignments

The first challenge we consider is how to allocate runs to nodes. Currently programmers manually decide which forecasts should run at each compute node. Each programmer typically has exclusive use of a subset of the nodes to

simplify workflow management, but nodes may be shared manually as needed. For example, a programmer in need of extra CPU cycles (e.g., to run a special forecast or because another node is down) who observes an idle node may email other programmers and ask permission to temporarily borrow the node. In the existing system there is no easy way to see the “big picture” of where and when workflows are running and which nodes are idle, and no easy way to estimate the likely effects of moving a workflow to a different node. Seeing the big picture is also useful to evaluate hypothetical scenarios, e.g., anticipating hardware needs as the number of forecasts grows.

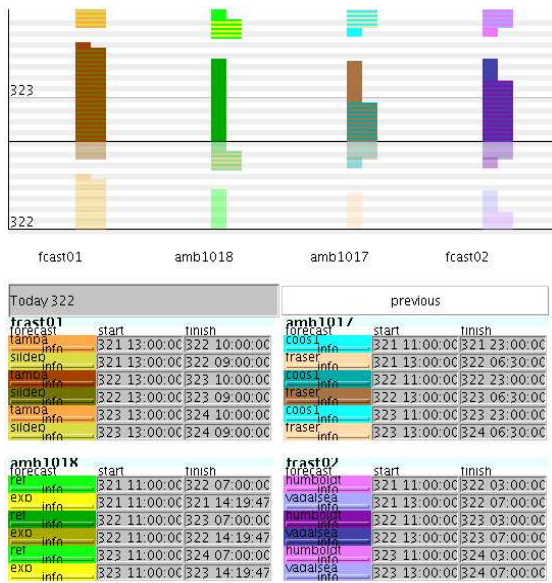


Figure 3. ForeMan: forecast management interface

Interface Figure 3 shows part of ForeMan, a tool we have implemented to manage allocation of forecast workflows to nodes. Note that this figure only displays a subset of all runs, due to space limitations. The interface displays both currently executing forecasts and those scheduled to run in the near future, and thus provides a centralized view of the current state of the factory. We describe some of its important features below.

The top half the figure monitors forecast execution. Each rectangle represents a forecast run. In this example, all nodes have two CPUs. Narrow solid-colored rectangles denote time periods where only a single forecast is running and a single CPU is in use, and wide multi-colored rectangles denote periods where both CPUs are being used concurrently by two or more forecast runs.

The black line shows the current time. All rectangles below the line are shaded out to denote forecasts that have fully or partially completed. Rectangles completely above the line (i.e., forecasts that have not yet launched) may be moved to other nodes or their starting times may be adjusted. The tool will automatically recompute the expected completion times of all affected workflows. Once an acceptable assignment of workflows to nodes is found, the user can click an “accept” button and the back end will automatically generate the needed scripts and commands. The back end can be tailored to any underlying scheduler or resource manager.

Implementation ForeMan uses a database of statistics (described in Section 4.3.2) gathered from prior forecast runs to estimate forecast running times and approximate an optimal assignment of forecasts to nodes. By default, each forecast is assigned to the node where it ran on the previous day. However, due to frequent changes in code versions, mesh versions, timestep granularities, and node availability, the previous day’s configuration may be suboptimal. Users can easily move workflows to different nodes using ForeMan, without making any changes to the underlying scripts that control forecast execution and data staging.

Currently ForeMan estimates the forecast running time by using data from past forecast runs. Users may manually adjust this estimate using workflow-specific knowledge, as described in Section 4.3.2. If a forecast is moved to a faster or slower node, ForeMan will scale the expected running time of the forecast by the relative node speed. In addition, we assume that a forecast consumes no more than one CPU at a time, i.e., serial execution¹. If the number of forecasts scheduled to run concurrently at a node is greater than the number of CPUs, we assume the available CPU cycles are divided evenly among the runs. For example, if three forecasts run concurrently on a node with two CPUs, ForeMan will compute the expected completion time of each assuming each forecast gets 2/3 of the available CPU cycles. We have validated this assumption empirically using data from past forecast runs, for a relatively small number of forecasts. We are planning to incorporate more sophisticated performance modeling techniques as part of our ongoing work.

We note that many existing cluster scheduling tools may not explicitly provide the above functionality, instead assuming that required resources (e.g., CPUs) cannot be shared with other users or tasks. However, in a factory setting, it is sometimes desirable to have multiple runs share CPUs, even though it will extend their individual completion times. For example, scientists may want early initial data products from multiple forecasts rather than restrict a CPU to a single run at a time.

¹If parallel forecasts are deployed in the future, we will extend ForeMan to support “mega-jobs” spanning multiple CPUs.

ForeMan can approximate an optimal assignment of workflows to available nodes, using bin-packing heuristics [4] and periodic scheduling techniques. Users can manually adjust these assignments as needed. ForeMan also allows users to prioritize forecasts, and may automatically delay or drop lower priority forecasts if needed (e.g., if data arrival is delayed or a node fails). A challenge in this setting is determining which forecasts to reschedule when reassigning forecasts. For example, when a new forecast or node is permanently added to the factory, rescheduling all forecasts may be beneficial, but when a node temporarily fails users may wish to reschedule only a subset of forecasts. Refining our scheduling algorithms and policies to handle these complexities is an area of ongoing work.

4.2 Data Flow

Another challenge in managing the forecast factory is determining architectures for data generation and movement. Currently the factory executes all tasks in a single forecast run at a single node. As model outputs and data products are generated at the compute node, they are periodically copied to a central server where they are made available to data product users. We present this architecture in Figure 4. In this architecture a simulation incrementally generates model output files, and a script `master_process.pl` launches tasks that incrementally compute data products from the model outputs. Finally, another script runs `rsync` in the background to incrementally copy completed portions of model outputs and data products to the server.

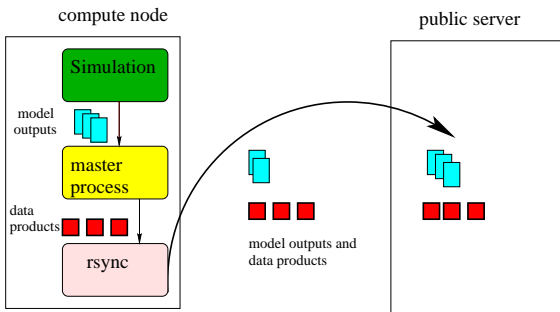


Figure 4. Architecture 1: Generating model outputs and data products at client

While straightforward to implement and launch, this architecture has several limitations. First, the simulation and data product generation tasks both consume considerable amounts of memory and CPU cycles, so running them concurrently may increase the running times of both. Also, both model outputs and data products must be copied to a remote server after they are generated, which consumes bandwidth.

Since the CPU at the public server is lightly used, we considered an alternate architecture, shown in Figure 5. In this architecture, the simulation model runs at a compute node along with an `rsync` script that incrementally copies model outputs to the server. The `master_process` script launches tasks to generate data products at the server. This architecture reduces bandwidth consumption because data products do not need to be moved to the server. It also exploits available server CPU cycles.

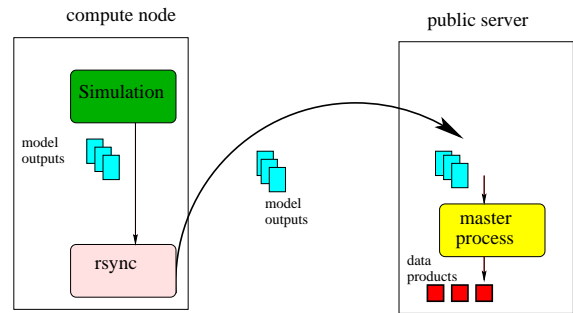


Figure 5. Architecture 2: Generating data products at a server

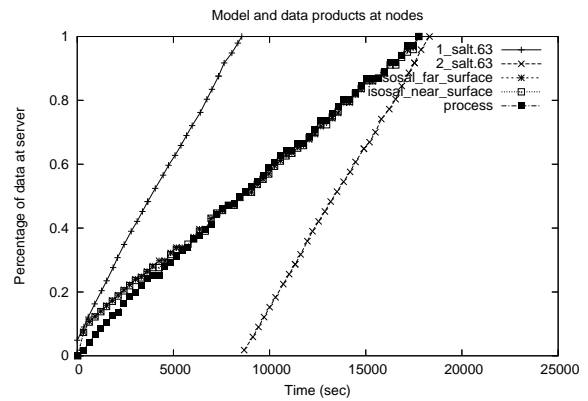


Figure 6. Time until all data appears at server for Architecture 1

We evaluated the performance of both architectures on one of the existing forecasts using two compute nodes connected by a local area network. The “server” node ran CentOS and had 1GB of RAM and a speed of 2.60 GHz. The “client” node ran Fedora Core 1, and had 1GB of RAM and a speed of 2.80 GHz. We ran the ELCIRC simulation [18]. Figure 6 plots the percentage of data (from selected

model output files and data product directories) resident at the server as a function of time. The files `1_salt.63` and `2_salt.63` are two of the model outputs generated by the ELCIRC simulation program, representing the salinity of the water on days 1 and 2 of the forecast period. Results for other model output files were comparable. The directories `isosal_far_surface`, `isosal_near_surface`, and `process` contain some of the data products generated from the model output.

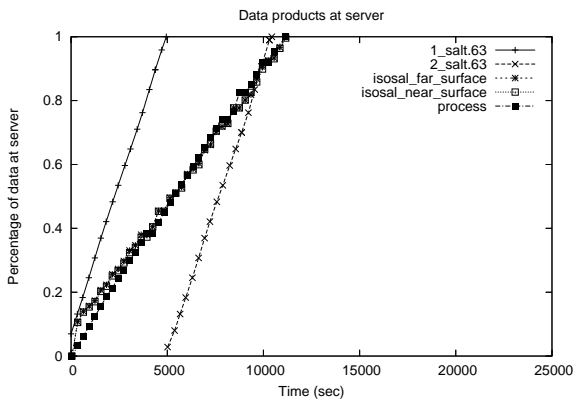


Figure 7. Time until all data appears at server for Architecture 2

Figure 7 plots the percentage of data at the server for the same model outputs and data products, using the architecture presented in Figure 5. As hypothesized, the end-to-end time to generate all required data and move it to the server is significantly reduced when the model run and data product generation tasks are done at separate nodes. Running all tasks at a single node has an end-to-end time of about 18,000 seconds (5 hours), while running the simulation model and data product generation at separate nodes takes about 11,000 seconds (around 3 hours). We observe that in Figure 6 the final model outputs and data products arrive at the server at around the same time, while in Figure 7 the final data products appear slightly later, due to the extra time needed to generate the final set of data products at the server after the final model outputs are copied there. However, overall product data becomes available at the server much earlier in the second approach. For many forecasts, data products account for as much as 20% of all data generated in a run. Thus, this architecture could significantly reduce bandwidth consumption.

We briefly discuss scalability issues. While simulations run continuously and thus consume nearly all available CPU cycles, tasks to generate the data products are launched repeatedly as additional simulation model output is generated.

Thus, the server CPU may be idle between increments of model outputs. We have evaluated the performance of Architecture 2 generating four sets of data products concurrently at a server and found that running these four sets of tasks concurrently increases the completion time by only a small amount (about 3000 seconds). We are investigating architectures to scale to a larger number of forecasts.

4.3 Log Data

Analyzing log data can show the effects of changes to a forecast on its running time, and can identify anomalies and long-term trends. Even minor modifications may change a forecast's running time by several hours, and may require changing node assignments or starting times of some forecasts. Thus, understanding the effects of changes on run times can improve management of the forecast factory, both via better accuracy of estimates and by identifying changes that are likely to significantly affect forecast run times and hence require reallocating resources. We first present observations from actual log files, which show the benefits of using this data, then discuss in greater detail how we extract and leverage log data in our forecast factory.

4.3.1 Examples

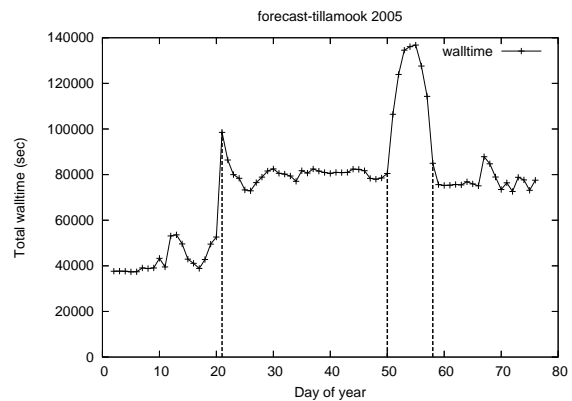


Figure 8. Effects of timestep changes and addition of new runs on the Tillamook forecast

We present wall-clock times of two different forecasts as a function of day in Figures 8 and 9. Figure 8 plots the running time of the Tillamook forecast for days 1–76 of 2005. On day 21, the number of timesteps in the forecast was doubled from 5760 to 11520, and we observe a corresponding increase in the running time, i.e., it approximately doubles from around 40,000 seconds to 80,000 seconds. We

note that no changes to the code version or mesh occurred during the period plotted in the figure.

Between days 50 and 60 we observe a significant increase in running time that has a “hump” shape. We believe that the increase was caused by the addition of several new forecasts to the system around day 50, some of which may have interfered with the execution of the Tillamook forecast. We attribute the “hump” shape of the graph to a cascading effect. On day 50, the forecast took about 100,000 seconds to complete instead of 80,000 seconds. Note that one day is 86,400 seconds and forecasts generally start at the same time each day. Thus, the forecast of day 51 began at the same node as day 50, before day 50 had completed. Since the two forecasts were competing for CPU cycles, the completion of the day 51 forecast was delayed further, and so on. After a couple days the run time begins to stabilize, most likely due to moving other forecasts off of this node, and the wall clock time returns to its earlier levels.

The hump in this graph is an example of a phenomenon that cannot be easily observed through log files alone, but provides a valuable lesson to scientists trying to incorporate new forecasts into their factory. We believe that our database of forecast log data will significantly improve forecast factory management.

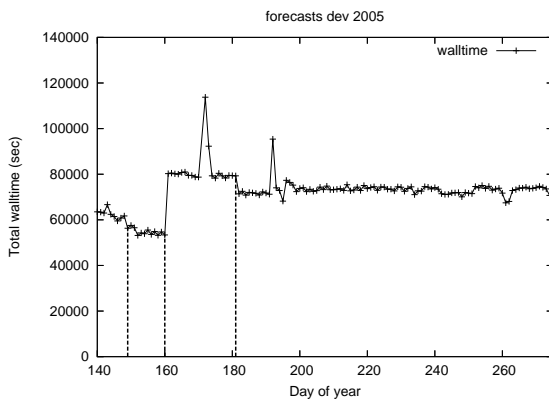


Figure 9. Effects of code changes and mesh changes on the dev forecast

As a second example, consider the developmental forecast (dev) for days 140–270 of 2005 shown in Figure 9. The dev forecast is continually being adapted, so new code versions are common. Around day 150 we observe changes to both mesh size and code version, causing running times to decrease by about 5000 seconds (about 1.5 hours). Around day 160 we observe a significant increase in the running time of the forecast due to a major version change in the simulation code. The running time increased by over 26,000

seconds, or over 7 hours. Around day 180 we observe a decrease in running time of about 7000 seconds (almost 2 hours), again due to a code version change. We observe several smaller changes as well. We note two spikes on days 172 and 192; we suspect these were caused by contention with other forecasts for CPU cycles. This graph shows that changes to both mesh size and code versions may significantly affect a forecast running time, so estimates of forecast running times and forecast node assignments may need to be altered whenever such changes occur.

4.3.2 Collecting and Using Log Data

In the forecast factory each forecast runs in its own directory, where all executables, inputs, outputs, and log files are stored. This flat directory structure made it difficult to observe changes to forecasts over time, such as those shown in the previous section. It also did not allow queries such as “find all forecasts that use code version X”. To address this limitation of the directory structure, we populated a relational database with statistics extracted from forecast directories. We used scripts to crawl all existing directories to parse log files and gather other relevant information, e.g., code versions. We note that since the database contains tuples for each *run* execution (one per day, for each forecast) rather than for each *task* execution (possibly thousands per day, for each forecast), the overall size of the database is relatively small.

Using Statistics for Estimation We briefly discuss how statistics from our database can be used to estimate forecast running times. We first consider timestep granularity. As observed in Figure 8, forecast running times appear linearly proportional to the number of timesteps. Thus, when users change the timestep granularity of a forecast, they can query the database for earlier runs of the forecast and scale the running time accordingly. We have also observed a near-linear relationship of run time with the number of sides in a mesh. Estimating the effect of a mesh change on run time is sometimes non-trivial because other changes to a mesh (e.g., depth of a region) may also affect run times, and because mesh changes often accompany code version changes. However, all other things being equal, an increase in the number of sides will generally increase the running time proportionally, so scaling results from earlier runs after a mesh change can provide an approximation.

Estimating the effect of a code version change on forecast running times is more difficult to automate due to the complexity of the underlying simulation models. However, the CORIE programmers have years of experience developing forecast simulations and may have some intuition as to how their code changes will affect forecast running times. For example, a programmer may estimate that a new code

version will run 10% faster than the previous version. Using statistics from a prior run on a similar node and with similar mesh size and timestep granularity, the programmer can obtain a rough estimate of the running time of the new code version on a particular forecast.

Database maintenance Keeping the database up-to-date is important to monitor the progress of currently executing forecasts. Once a forecast has completed, its statistics do not change. However, a currently executing forecast will have incomplete statistics in the database. For example, it does not have a completion time. In our existing implementation we populate and maintain the database by running daily Perl scripts that crawl new forecast directories and extract all relevant statistics. However, periodically crawling directories does not provide the most up-to-date statistics for currently executing forecasts, which limits the effectiveness of our interface for monitoring forecast execution. Executing scripts more frequently will improve the freshness of the database, but may have a high overhead.

We are considering other means to keep the database up to date. One option is to insert commands into the run scripts to update the database. Automatically updating the database would provide several key benefits. For example, it would eliminate the need for background scripts, and would ensure that statistics on currently running forecasts are accurate. Finally, it would allow us to benefit from some existing database technologies. As observed by others [11, 13], databases can improve scientific workflow management because they have functionalities such as triggers, transactions, and concurrency control that are useful for scientific workflows. However, using a database to directly manage the forecast factory may be impractical due to the large number of tasks in a single forecast and the extensive overhead of modifying existing workflow code. Incorporating database commands into existing forecast runs would allow the forecast factory to leverage some benefits of database technology without requiring extensive modifications to the workflows.

5 Conclusions

Data product factory management requires efficient architectures, planning, and scheduling to maximize use of available resources while ensuring timely completion of all workflows. While recently there has been a considerable amount of research in scientific workflow management, little attention has been given to these specific challenges posed by data product factories. In this paper we have discussed how we have addressed these challenges in the CORIE forecast factory. Many similar problems arise in other product generation systems, thus, we believe many of our solutions apply to other domains. In future work,

we are investigating how to incorporate made-to-order (on-demand) products into the system along with the made-to-stock products currently manufactured in the factory. We are planning to deploy ForeMan, our factory management interface, for use by CORIE scientists in the near future.

6 Acknowledgments

We thank Paul Turner and the rest of the CORIE team for providing helpful information and feedback. This research is supported by NSF grant CCF-0121475.

References

- [1] I. Altintas et al. A framework for the design and reuse of grid WorkFlows. *Proc. 1st Workshop on Scientific Applications and Grid Computing (SAG)*, 2004.
- [2] A. Baptista, M. Wilkin, P. Pearson, P. Turner, and P. Barrett. Coastal and estuarine forecast systems: A multi-purpose infrastructure for the Columbia River. *Earth System Monitor, NOAA*, 9(3), 1999.
- [3] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. *Proc. HPC*, 2000.
- [4] E. G. Coffman, M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1), 1978.
- [5] K. Cooper et al. New grid scheduling and rescheduling methods in the GrADS project. *Proc. IPDPS*, 2004.
- [6] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Workflow management in GriPhyN. *The Grid Resource Management*, 2003.
- [7] E. Deelman et al. Pegasus: Mapping scientific workflows onto the grid. *Proc. 2nd AcrossGrids Conf.*, 2004.
- [8] I. Foster, J. Vockler, Michael Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. *Proc. SSDBM*, 2002.
- [9] W. Hopp and M. Spearman. *Factory Physics: Foundations of Manufacturing Management*. Irwin/McGraw-Hill, 1996.
- [10] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. *Proc. ICDCS*, 1988.
- [11] D. Liu and M. Franklin. GridDB: A data-centric overlay for scientific grids. *Proc. VLDB*, 2004.
- [12] T. Oinn et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(7), 2004.
- [13] S. Shankar, A. Kini, D. DeWitt, and J. Naughton. Integrating databases and workflow systems. *SIGMOD Record*, 34(3), 2005.
- [14] A. Szalay, P. Kunszt, A. Thakar, J. Gray, and D. Slutz. The Sloan Digital Sky Survey and its Archive. *ASP Conf. Ser., Vol 216, Astronomical Data Analysis Software and Systems*, 2000.
- [15] Torque/Maui Cluster Scheduler and Resource Manager. <http://www.clusterresources.com>.
- [16] M. Wiczeorek, R. Prodan, and T. Fahringer. Scheduling of Scientific Workflows in the ASKALON Grid Environment. *SIGMOD Record*, 34(3), 2005.
- [17] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record*, 34(3), 2005.
- [18] Y. Zhang, A. Baptista, and E. Myers. A cross-scale model for 3d baroclinic circulation in estuary-plume-shelf systems: I. formulation and skill assessment. *Cont. Shelf Res.*, (accepted for publication).