Ultra-large docking.
How to run ultra-large GOLD docking jobs on cloud resources.
1. Introduction

For almost three decades, computational chemists have contributed to the hunt for new molecules in a drug discovery context, and *in silico* approaches are now firmly embedded within pharmaceutical and agrochemical companies.

One of the computational methods available to scientists, when the structure of the protein target associated with the disease is known, is protein-ligand docking: a virtual library of compounds is docked into the binding site of the protein, in order to rank and prioritise compounds by their likelihood of binding.

With advancements in cloud computing, and the availability of large and high-quality datasets of potential ligands, it is now possible to perform such studies at ultra-high-throughput scale whilst incurring only modest costs.

The ability to generate billions of hypothetical molecules *in silico* and then reliably synthesise molecules to order in a matter of weeks, for example as offered by Enamine Ltd., has created a different paradigm for structure-based drug design. Discovery scientists have a clear need to be able to select and screen *in silico* some of these billions of hypothetical molecules, and then select the best candidates to synthesise and screen, in order to identify leads for a new target. Being able to screen the larger molecular spaces enumerated by such vast libraries may therefore result in identifying novel compounds with high activity and selectivity.

There are specific challenges around data output & speed that need to be addressed to enable ultra-large docking of such libraries. First, the volume of data generated by these virtual runs must be manageable. Typical docking runs generate a lot of output for each molecule, so scaling to docking billions of compounds requires the judicious storing of only the minimal data required to enable the prioritisation of compounds for more investigation after an ultra-large docking screen. Second, the timescale of these virtual runs must be convenient for structure-based drug design programs.
The large number of dockings required can be achieved fast enough by parallelising docking runs across many cores and making use of the seemingly infinite CPU access provision from cloud computing providers.

The most recent developments of the GOLD docking software have allowed us to satisfy these two key requirements of the ultra-large docking case, and to demonstrate the robustness of this implementation by an exemplar docking run of about 130 million compounds against a chosen protein structure, using GOLD deployed onto cloud-based computing resources.

When doing structure-based virtual screens, there are two additional challenges that a computational chemist will need to give thought to. The first key aspect is the preparation of compound libraries for docking. GOLD requires that compounds to be docked are supplied as 3D conformers, with explicit protonation and tautomer state, as these are not varied during docking. Therefore, it is important to determine the most likely protomer/tautomer for each compound; if this is not possible, then supplying each compound as several possible protomers/tautomers is also viable. In terms of the 3D conformation of each compound, GOLD will only modify the torsion angles of rotatable bonds (and the conformation of flexible rings, if this option is enabled) when exploring the ligand’s conformational space during a docking run, but not the bond lengths or valence angles so these must be reasonable. The second key aspect is how to analyse and visualise the screening results.

2. Containerisation, Kubernetes and RabbitMQ

Traditional use of cloud resources has relied on deployment of virtual machines into the cloud, but such approaches have some drawbacks. Virtual machines (VMs) are rather heavyweight as they include an entire operating system. This means that software installed on each VM must account for the chosen operating system with all the possible drawbacks that can be experienced. Further, the size of each VM can make such approaches expensive.
Containerisation using Docker containers resolves many of these issues as the application is wrapped up into a single self-contained entity that is insulated from the parent operating system. The container itself is relatively small, containing just the software and its dependencies, and can run on any suitable computing environment.

Kubernetes is an example of a system that allows orchestration of docker containers on a cloud computing service. The common cloud services (Azure, Amazon Web Services (AWS), Google Cloud Platform (GCP)) support using Kubernetes. Finally, RabbitMQ is a message passing system that can be used in a Kubernetes cluster to facilitate queuing of jobs and harvesting of results. It is deployed as a Docker container itself.

Figure 1. Information flow through the system.

The CCDC cloud computing infrastructure for docking using GOLD is implemented using a GOLD Docker image which can be easily deployed onto a Kubernetes cluster.
**System Requirements**

As outlined in Figure 1, the user’s institute will need to provide 3 components.

Firstly, the institute will need to create a suitable Kubernetes cluster. Such architectures are supported on many common cloud platforms including; Azure, AWS and GCP.

We recommend that such a cluster is deployed with the help of your local IT support. Instructions on setting up, scaling and managing a Kubernetes cluster are given here.

You will also require a local computer to act as a local license server for the GOLD workers in your Kubernetes cluster; the server should serve an unlimited license to GOLD with a suitable lease duration as available through CSD-Enterprise and CSD-Discovery.

Finally, in order to run the system, the user will need a local machine that runs the submit and harvest mechanism scripts.

**System Requirements**

The central component of cloud docking using GOLD is provided by a Docker image in CCDC’s Harbor. This image implements a simple environment in which GOLD can run, taking a set of standard GOLD input files.

It is used in tandem with a ‘standard’ RabbitMQ Docker image which is responsible for basic job scheduling and result harvesting.
The Kubernetes Cluster

When using the GOLD Docker image, we use a Kubernetes cluster on the cloud. This cluster consists of one RabbitMQ scheduling pod and many worker pods. The queuing system’s role is to schedule the dockings provided in a balanced way to possibly many 1,000s of docking pods. The system has been tested when scaled to use 1,300 GOLD docking pods in tandem with a single queue scheduler.

To achieve this, we used a cluster that consisted of 99 F16s_v2 nodes. These are high end compute optimised virtual machines with 16 virtual CPU cores using hyper-threading, 32 GB of memory and 256 GB of temporary storage space. Consequently, each node has the computational bandwidth to run many dockings at once: In theory, a cluster running 1,584 scheduled jobs in parallel, each job typically docking a batch of 2,000 ligands would operate at full CPU. In practice we used 1,300 docking pods in our experiment: This was in response to the Kubernetes best practice to aim at around 80% CPU usage on the cluster.

Example Scripts

The CCDC provides two example scripts that manage the process of submitting and harvesting the results. The submit_tasks.py and collect_results.py are scripts that show how you can interact with the RabbitMQ queue. In this implementation, the expectation is that the user will run the collect_results.py script continuously to collect and sort results from the RabbitMQ results queue and write them locally to disk. Users can tailor these scripts to their own specific needs.

User documentation

User documentation for the nodes is provided here. The documentation explains the expected steps to configure the system once you have a Kubernetes cluster configured.
3. A large-scale docking benchmarking experiment

The ultra-large virtual compound library was created by downloading a set of 3D protonated molecules from ZINC, selected to be in the drug-like compound space, with a logP between 2 and 3, and a molecular weight between 300 and 350 Daltons (the ZINC Goldilocks set). We picked approximately 126 million structures from this set for this experiment.

The ultra-large library was screened against Prostaglandin-endoperoxide synthase 2 protein, also known as COX-2. The inhibitors of COX-2 inhibit the conversion of arachidonic acid to precursors of prostaglandin giving them both analgesic and anti-inflammatory properties, and so COX-2 inhibitors are implicated as a target for several disorders including pain relief. Many non-steroidal anti-inflammatory drugs have been developed that target COX-2, including Celebrex which, in 2011, was one of Pfizer’s largest selling drugs (see drugs.com). COX-2 inhibitors have also been researched for anti-cancer activity linked to inflammation. The specific COX-2 protein was prepared from the publicly available 1CX2 PDB structure; this is the COX-2 target from the Directory of Useful Decoy (DUD) dataset.

The protein structure for the GOLD calculation was processed by protonating the 1CX2 structure for COX-2, and then visually inspecting the target to ensure that the side-chain orientation of all glutamine, asparagine and histidine residues were correct, as were the protonation state of all histidine residues. All water molecules were removed from the protein structure. The binding site was defined by selecting all atoms within 6 Å of the ligand of the 1CX2 crystal structure. The resulting cavity atoms are highlighted in green in the prepared 1CX2 protein (see Figure 2 overleaf).
The scoring function was chosen as ChemPLP, because we had found in 2012 [1] that this GOLD scoring function provides overall the best performance against the set of 40 DUD targets, compared to the other available three scoring functions. The genetic algorithm parameter settings were set to automatic (ligand-dependent) and with search efficiency (autoscale) set to 0.1, which corresponds to library screening. In the run-up to our latest ultra-large docking experiment, we had found that the performance we reported in our 2012 publication [1] (where autoscale was 0.3) was not significantly affected by decreasing the search efficiency to 0.1. This lower search efficiency was therefore chosen as it would provide us with the fastest speeds we needed for the ultra-large docking case. All other settings remained as the GOLD defaults.

Each ZINC compound was docked 5 times into the binding site (with early termination switched off), and only the best binding pose score details were kept by using the MIN_OUT keyword. This means all the score terms for all the docked structures were retained. We then post-processed the results to select the highest-ranking compounds and assess the enrichment against known actives.
This work is primarily an exercise in deployment and retrieval of a large scale run with a focus on learning about scaling to large volumes of ligands. However, as a matter of academic interest, we decided to evaluate whether we would have enriched retrieval of structures that would be of increased interest as COX-2 inhibitors. The ZINC database contains activity subsets where a compound has some known implication against a given gene target, in this case PTSG2, so we downloaded and prepared all 3419 molecules in this subset then seeded them into the full set of compounds so we could assess the retrieval enrichment of this set. We note that this is an extreme challenge for an enrichment study as compared to a more traditional active and decoy test due to the sheer volume of molecules being screened.

The aim was to dock more than 100,000,000 of the downloaded ZINC set. The Kubernetes cluster was deployed on Azure (US East) system with 99 F16s_v2 nodes. 126,345,649 substances of the ZINC subset were picked for docking. In total this meant 631,728,245 dockings as each compound was docked 5 times. The submission script was set to batch the ligands into batches of 2,000 ligands each.

**Outcome**

The run took 32.8 hours in elapsed time to deploy the cluster, dock all the ligands into the target and shut down the cluster using screening docking settings. This equated to approximately 3.85M molecules per hour.

**Performance**

The performance of the docking run was consistent for the duration of the run while the cluster was fully provisioned. A summary of the per-ligand time taken is given in Table 1.

<table>
<thead>
<tr>
<th>Table 1. Ligand specific metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average Elapsed time per ligand (seconds)</strong></td>
</tr>
<tr>
<td><strong>Median Elapsed time per ligand (seconds)</strong></td>
</tr>
<tr>
<td><strong>Maximum time for a ligand (seconds)</strong></td>
</tr>
</tbody>
</table>

www.ccdc.cam.ac.uk
In early trial runs, we noted some degradation of performance if the RabbitMQ nodes for queuing of jobs and result retrieval were not sufficiently provisioned. This could lead to worker nodes lying idle. We would therefore recommend that these nodes are given sufficient resources to avoid this. More information on provisioning of RabbitMQ is available here.

While in principle the performance was satisfactory, we did note some performance degradation when scaling to a very large run. Initially with a smaller number of nodes, we achieved quicker speeds per ligand, but as we scaled to 1300 nodes there was a small increase in time taken per ligand when using the largest cluster setup as compared to a smaller cluster. Similarly, as jobs completed towards the end of the run and no more were scheduled, docking speed increased slightly. This is shown below in Figure 3. The timings are consistent over the run with the exception towards the end. This relates to the nature of the final tranche that was run, which contained smaller more rigid ligands than in the earlier tranches. The autoscaling in GOLD means these ligands complete more quickly.

Figure 3.
Time per ligand in batches as a function of batch completion time
Security

Docker containers and RabbitMQ are inherently quite secure. That said, one area where additional security can be added is in the communication between the submit_tasks.py, collect_results.py scripts and the RabbitMQ system. We note that enabling security around the GOLD-HPC system ideally required specialist IT knowledge of the techniques used. We recommend that users work with their local IT support if they wish to add higher levels of security. The options here are not specific to GOLD-HPC per-se, rather methods that can strengthen security in general when working with large scale cloud systems scheduled through RabbitMQ.

We identified 3 possible methods for securing job communication. All three are independent of each other so an end user can choose to use any number of them together, though Method 1 would probably render Method 2 redundant.

Method 1. Run the entire job including scheduling and data harvesting inside the cloud system

In this method, the user configures a separate Docker pod to act as the machine running the submission and harvesting scripts. The pod must be provisioned with sufficient CPU and virtual disk space to store the input data and to process and store the outputs from the job (virtual storage would ideally be a separate device and configured for use by the pod). This has the advantage that the entire system including the submission and harvesting scripts then can reside behind one single firewall with the security this offers. The user then needs to transfer the information garnered from the cloud back to their own systems at the end of the run: this can be achieved using standard secure copying techniques (such as scopy) independently of the kubernetes system.

The disadvantage of this method is that, dependent on the choice of cloud provider, it may increase the cost due to the need for on-cloud data storage and large-scale information transfer between the cloud provider and the user’s local systems. In effect the data transfer and storage costs double in this scenario as the inputs and outputs must be transferred from the kubernetes system to the separate pod and then from this pod back to the user’s own internal systems. Further the input data (in this experiment this amounted to 0.1 TBytes) would reside on cloud system storage.
Method 2. Port-forwarding
We explored port-forwarding from a local machine to the kubernetes cluster. This means the kubernetes cluster can be hidden behind a firewall preventing external access to any of the cross-cloud communications but the submission and harvesting scripts (and the input data and results) reside on a local machine. The port-forwarding means that only this local machine can communicate with the kubernetes cluster and data is transferred securely. In effect the port forwarding means the local machine is pulled inside the firewall surrounding the kubernetes cluster.

In our experiments port forwarding could be included in the configuration, but when running at scale we found that we could experience periodic timeouts due to lost connections. This is a seemingly rare but endemic problem with this method that is not easy to solve in a robust manner. This is not a fundamental problem with kubernetes or RabbitMQ, rather a problem with network connectivity in general with a long-lived long-range connection (in this case from the United Kingdom to the US East Coast).

One suggestion to prevent problems with lost connections is to have an independent task whose sole purpose was to immediately restart a broken connection when detected. Should a user wish to use port forwarding they may choose to do this independently of the GOLD-HPC scripts by polling the connection every 10 seconds or so and using kubectl to restart if required.

Method 3. Communication Encryption
The scripts for submission and retrieval of data from the scheduling and results queues use the python pika library (v1.1.0) to implement communications, as can the layers of communication between the Kubernetes queuing system and the worker pods.

These can have TLS encryption layered in should an end user wish to have the data transfer encrypted. An example of using pika, RabbitMQ and TLS encryption is given here.
The end user will need to have set up and use suitable encryption certificates to enable TLS encryption as outline in the RabbitMQ documentation here.

This was our method of choice in our large-scale experiment and we conclude that this works robustly at scale. This is now the default in the 2020.2 release.

**Cost**

The total computational cost using the configuration described in the experiment amounted to £2078 GBP (as of August 2020) for running these large-scale dockings. In total we docked 126,345,649 molecules from the ZINC set using a quick docking protocol. We recommend, when using GOLD-HPC, first trialling your docking configuration locally to explore the relative speed of the configuration in use (and if possible its expected efficacy) if you plan to run a large-scale docking, as the speed for a given system will vary dependent on several factors, namely.

- Binding site size and nature
- Size of the underlying ligands
- Choice of parameterisation of the underlying search

**Enrichment**

While the purpose of our experiment was to explore the technical feasibility of very high-scale dockings on a large data set, and identify the challenges this poses, one interesting element is to examine the enrichment of known active molecules in the results returned. As noted above, we seeded 3,419 molecules with known activity against the gene PSTG2 as annotated in the ZINC database (actives with PTSG1 activity were omitted, even though these compounds often show cross-reactivity between these two related systems). Using the results in this study, we can assess the early enrichment. In a docking study of this nature, we would probably expect the end user to take the top 1,000,000 molecules and subject them to more scrutiny: for example, redocking and retaining the best pose then perhaps re-scoring those pose using alternative methods to filter the results further.
Note that in this experiment, if the scoring function causes no enrichment, we would expect only a single active in the first 29,239 structures. However, we know from previous experiments that GOLD is effective at active retrieval on this target and model system[1]. Thus, for this experiment we were interested to see the numbers of actives that were retrieved in the highest 0.0001% (top 127) to the highest 1% (top 1,000,000) of the database. The results are shown below in Table 2.

<table>
<thead>
<tr>
<th>Active count (of 3419)</th>
<th>Enrichment Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 0.00001% (Top 13)</td>
<td>3</td>
</tr>
<tr>
<td>Top 0.0001% (Top 127)</td>
<td>22</td>
</tr>
<tr>
<td>Top 0.001% (Top 1264)</td>
<td>102</td>
</tr>
<tr>
<td>Top 0.01% (Top 12635)</td>
<td>275</td>
</tr>
<tr>
<td>Top 0.1% (Top 126346)</td>
<td>578</td>
</tr>
<tr>
<td>Top 1% (Top 1263457)</td>
<td>1063</td>
</tr>
</tbody>
</table>

There are, of course, in these results many confounding factors. Firstly, we’ve selected a target system that we know well. We have prior evidence that GOLD scores this system well. Secondly, we have not balanced the 126M compounds to have a similar chemical profile to the actives; for a strict evaluation of enrichment this would be necessary. This undoubtedly leads to significant artificial enrichment in these results overall.

That said, we assume that none of the 126M ZINC structures docked in addition to the actives are active themselves. In a real study some of the high ranked compounds found may indeed be active against the target. We note, for example, that some of the high ranked misses are known PSTG1 inhibitors according to ZINC; a highly related system. These results serve an illustrative purpose: we can conclude that if we have a discriminatory score for a given target and a good model system, we have a reasonable chance of finding active molecules relatively highly ranked in the list.
4. Conclusions

We have now created and tested a configuration that allows GOLD to be run on common cloud setups. Due to recent changes to the GOLD code, it is possible to minimize the volume of data generated and retained when docking, which is necessary for very high-scale runs over millions of molecules.

The system can be deployed at scale to cloud computing resources to virtual screening runs over very large libraries of 3D conformations using widely available cloud computing resources and relatively standard cloud computing methodologies.

References