Applied Research Lessons from CloudViews Project

Alekh Jindal

Gray Systems Lab
Microsoft

ABSTRACT

Industry research has a rich legacy in computer science [9]. However, as opposed to the blue-sky approach to research, increasingly there is a trend to align industry research more closely with the products. This is manifested in several new trends in industry research: (i) emphasis on product impact, e.g., improving existing products or seeing new ones coming around the bend, (ii) popularity of blended job functions, such as scientist, research scientist, and data scientist, and (iii) setting up research teams that are integrated within the product organization to forge closer collaborations. The latter is a case in point in Azure Data group at Microsoft, where the Gray Systems Lab (GSL) [2] is an applied research team within the product group. Such integrated research labs offer beautiful opportunities for combining research with product impact. Yet, due to their product focus from the get-go, applied research labs could also be challenging to get started. Fortunately, it turns out that there are a set of things that new researchers could do in order to set themselves up for success in a product group.

In this paper, we describe the key lessons learned from the CloudViews project [1] at GSL. CloudViews project started with identifying and reusing common subexpressions in big data workloads at Microsoft, however, it was also successful in spinning up a number of follow-up projects, establishing the GSL ties with the SCOPE[1] team, and seeding the bigger vision of workload optimization, resulting in the Peregrine [7] and Flock [4] projects. Although the lessons we discuss below are derived from the CloudViews project at GSL, we believe the learnings are applicable to other industry research settings as well. Note that there could be several successful ways of going about applied research, however, in this paper, we only discuss the things that we found useful in our experience from the CloudViews project.

[1]SCOPE [12] is the query processing engine in Cosmos, the big data infrastructure for running massive scale analytics, consisting of hundreds of thousands of machines and hundreds of thousands of analytical jobs, across the whole of Microsoft, including teams such as Bing, Office, Windows, XBox, etc.

1. GETTING STARTED

Researchers at Microsoft, and other places, often have the freedom to pick their projects. However, this could also be overwhelming given wide range of products and possible opportunities at a large company like Microsoft. Therefore, before getting started, it could be useful to familiarize oneself with some of the products and pick one that is promising and big enough, in terms of usage, revenue, etc. The choice may be narrowed down by talking to colleagues, managers, or potential business groups, and it is important to be open to learn new areas.

After picking a product, get ready for a long term investment with the product team. This means getting hands dirty by diving into the production codebases — often a leap for researchers who might be used to building standalone prototypes. Getting hands-on on production codebases is useful in keeping the research project grounded in practicality. Also, it is easy to get distracted with too many conversations across several products and there could be temptation to spin projects with each of them. However, it takes a lot of work to onboard and work with a product team, so an individual contributor might be better off by focusing on few things in the early stages. This also means to start sunsetting prior projects and collaborations from the graduate school.

To illustrate, the CloudViews project started with the intent to build connections with the SCOPE product team, in particular the SCOPE query optimizer team. There was a feeling that query optimization in cloud workloads is going to be an important area for long term investments from a research team, and the guidance from the leadership was to understand the SCOPE engine better, get hands dirty by going into the guts of the query optimizer, and explore opportunities for impact. This was a challenge since our prior experience was on building flexible storage systems and query optimizer was quite a shift. In summary, getting started in applied industry research at Microsoft involves taking a leap of faith on a product and preparing oneself for the long haul. Thinking of it as a strategic opportunity can certainly make things easier.

2. FINDING PROBLEMS

Research teams at Microsoft are typically separate from the product teams and so they often have the flexibility in picking problems. However, problem selection is an art and some people have a natural inclination for identifying big and interesting problems. Therefore, it is hard to come up with a recipe for finding the right problems. However, there are certain things one can do to increase their chances. A common pitfall is to jump to a solution, e.g., from one's previous work or from the state of the art, without fully understanding the problem. This often leads to solving something which is either not really a significant problem in the first place or solved in a manner that could not be applied in practice. While it is natural to try hard and polish the problem to fit to the prior solution, this could be very frustrating to the product teams. Instead, focus on understanding the real problems. Start with talking to the product teams, program managers, and customers if possible (e.g., if there are internal customers of the product). One may often end up with their laundry list of TODOs and bugs. One may be also shocked to see that many of the standard concepts and techniques from research papers have simply not made it to production. Don't get disheartened. Instead, consider this an opportunity to understand what works and what doesn't, why is it hard to get things right, and what are the core issues.

Once an engagement is established with a specific product team and there is some understanding of their open problems, the next step is to get access to their data and workloads to validate those problems. There is a tremendous value in being data-driven when looking for a research problem. It is also okay to shape a problem from the product team's TODO list, using ideas and intuition that are backed by data. The key is to convince the product team that the problem is worth solving. While navigating the problem space, loop back with the product teams, the PMs, and the customers, to share and corroborate the insights from their workloads. Ultimately, the project should answer positively to one of the following two questions: (i) will it improve an existing product or feature such that the customers could measure it, e.g., more efficient or more usable, or (ii) will it add new capabilities that allows the customers to do something more, e.g., enabling newer scenarios. The selected problem should also be feasible, i.e., estimate the work required early on. The goal should be to find a problem that is simple to get started, and yet has a big potential for impact.

Before arriving at the core problems to solve in CloudViews, we started talking to the SCOPE team and early conversations indeed resulted in a list of open bugs and issues that the SCOPE team needed help with. They also had a plethora of research work from the previous generation of team members that they were looking to transition into the product. These included optimizing recurring queries [3] and optimizing queries progressively [5]. Our observation was that while the primary focus was on optimizing each individual SCOPE query, there was a demand for reducing the overall operational costs, i.e., we could consider optimizing across queries as well. At the same time, there were existing pieces of infrastructure, e.g., plan identifiers called signatures [3] that were captured in the telemetry and could be used for multi-query optimization (MQO). These two observations led us to analyze the SCOPE workloads for MQO opportunities. Interestingly, we found common subexpressions to be a major problem in SCOPE workloads, where portions of the SCOPE queries were duplicated, thereby incurring redundant computations [8]. This early leg work went a long way in getting support from the SCOPE team.

3. GETTING FUNDED

A research team at Microsoft usually has limited resources, while taking a research idea all the way to a production ready solution requires a lot more effort[2]. Therefore, once there is an interesting problem that emerged from conversations with a product team and is backed by data, the next thing to think about is how to get the project funded. The first thing to do before expecting any funding is to align the problem with the product team's roadmap and priorities. A problem that is super interesting but does not fit into the product team's agenda is unlikely to get any funding. But wait what kind of funding are we talking about? The funding may consist engineers to help design, build, debug, test, and maintain your code, program managers to identify the market, make a business case, define priorities, and manage deliverables of your project, and customers to try out your solution, provide feedback, and finally to validate your success. In spite of this clear set of things that would be needed, the seed funding is always the researcher herself. This is simply because people are unlikely to to invest into a researcher's idea if she herself is not invested. So be prepared to be a one-person army in the beginning, and this is what we are going to focus on in the rest of this section.

Get things rolling by getting access to code, attending weekly meetings and scrums, and establishing a point of contact from the product team. Given the early stage of the project, be explicit about the fact that the point of contact is for lightweight consulting, e.g., one hour per week. Product teams are often driven by planning cadence and so it is unrealistic to ask too much from them

[2]The exception of course is if the project is completely research-oriented in nature without any plans for immediate product integration.

until things are put into the planning (otherwise the engineer's work may not end up being accounted). Digging into the code is important to understand the approaches to implement the key ideas. Large product teams often try a number of things, and the current idea could be one of them, may be in some other form. Learn about those attempts, their current status, and key learnings. It is important to know why things failed the last time they were tried in order to not make the same mistake. More importantly, see what pieces of code could be salvaged from those prior attempts. There is no point reinventing the wheel, so be open to building on top of prior art.

Once there is some understanding of the code and the prior art, the next step is to come up with an initial design. Often there is a big value in making the implementation iterative, e.g., thinking what could be the bare minimal version (v0) to demonstrate the ideas; what would be v1, v2, and so on. Identifying these versions helps stage the project into smaller achievable steps while also providing the bigger vision to the audience. Work closely with the engineering contact point to define the v0 and building a proof of concept (POC) around it. While this would be far from the production version, it is still important for understanding the system, getting a good sense of what pieces need to be built, and for earning early credits/credibility for being a doer.

Once there is a POC in place, pick a catchy project name. The good thing about names is that they can identify things succinctly, ideally in one word, rather than describing it in a sentence. The bad thing, however, is that it takes time for a name to stick in people's working memory. Although, this is exactly why it is important to pick a name early on and keep using it in everyday conversations, hoping for it to stick eventually.

In CloudViews, we jumped right into the SCOPE codebase to get a hang of things. While we got one person from the SCOPE team doing lightweight consulting, most of the driving and prototype building was on us. Given that CloudViews was the first MQO feature in SCOPE, we had to think through how to make it automatic in the SCOPE job service. We defined the v0 implementation and managed to get one pilot customer willing to try our feature if we were to build it. Once we got the POC working, we organized a demo, however, it crashed quickly since they had no cycles for offline data processing while our initial implementation would materialize common subexpressions in additional offline SCOPE jobs. This led us to think of materializing common subexpressions as part of query processing. We held on to this pilot customer for a long time and they were super useful in helping us understand the production scenarios. We also got a PM onboard to help productize the solution going forward. It turned out that PMs could be one of the best friends for the research teams and they should be actively leveraged for getting the customer perspective, driving product planning and priorities, interfacing product releases, and leading customer adoption. Finally, the goal for selecting the name of CloudViews project was to keep it simple, relatable, and consistent (last one being very important).

4. BUILDING COLLABORATIONS

Developers at Microsoft are part of the product teams and even after a problem has been identified that the product team agrees on, and a POC has been built that the PM team (and hopefully one early customer) is willing to buy, the researcher still needs to builds active collaborations with the developers in the product team. This is because typical funding for research projects at Microsoft does not result in a dedicated set of people, at least not for a fresh new researcher. Rather, the researcher needs to identify the key challenges and work items that need to be done to fully flesh out the POC, and rally people for their support. Of course, this would require back and forth discussions with the product teams. Remember the goal is to create a production ready feature/system and it needs to be conveyed as new work (and it should be new work), since nobody is interested in wrapping up a POC where all credits have been already claimed. While discussing, also consider who is the best person for each of the tasks and why. People could have different roles such as general interest, direction setting, brainstorming, consulting, planning, developing, customer interfacing, deployment, testing, etc. Try getting as many people onboard as possible for the long term, although everyone should have a clear well-defined role. Thereafter, talk to the managers and get approvals for the peoples' time. Work with the PMs to have the above plan added to the future planning cadence, so that people's time is officially accounted for.

One thing to note is that there is a value in creating a virtual team (also referred to as V-team) for the project. This could be as simple as creating an email distribution list or a SharePoint page. The goal is to give collaborators a sense of belonging and a means to communicate across a set of people working towards a common goal. Its good to be as inclusive as possible when adding people to the V-team. The entire process of discussing project implementation details with the product team and getting specific support from them by creating a V-team could go a long way in building sustained collaborations, even beyond the current project.

There was a long code review cycle for the initial CloudViews POC. We got people from different SCOPE component teams involved. Initially, we pushed for dedicated resources, but then we divided our ask into fine-grained tasks and convinced the component teams to add them to their planning with people hours assigned. The

CloudViews V-team had people from different component teams, with 8 members at its peak. We created mailing lists, held regular scrums, and even organized team events for the V-team. This was also a good time for us to present the problem statement, the POC, and the collaborations to the leadership team.

5. WALKING THE TALK

By now a lot of ground work has already been done, however, now comes the most important phase of the project — the execution phase. All the planning and preparation could go to waste if things are not executed properly. So now is the time for hands down execution using all the people and resources that have been gathered. Remember the project is now visible and people have committed their time to it, so one needs to be very diligent at how they go about things. Also, remember that the V-team was created for a reason, so delegate tasks to people in the V-team. Assign people, including the researchers, ownership of smaller pieces and make them accountable for it. Organize daily scrums in the beginning so that everyone is clear about their tasks and is able to make/report progress. The goal should be to orchestrate the execution and making sure people are unblocked as quickly as possible. At the same time, it is important to be hands on and not lose touch from details. Since the founding researchers have the best understanding of the project, they need to remain on top of things for a while. As the project progresses, there might be possible conflicts amongst V-team members: people not agreeing on the design or the APIs, not coordinating the action items, or loosing interest in the project. Overcoming these conflicts and bringing people on the same page is a crucial people skill to develop. All along, it is important to listen to diverse opinions, even though one may or may not agree.

Rarely a product is developed in a single shot, therefore be constantly reminded on applying an iterative model, i.e., building versions v1, v2, v3, and so on. Test the feature end-to-end as early as possible, preferably in production environments, to iron out the bugs or correct design decisions. All along, keep people informed about the progress. Once the productized version (v1), works, present it to the customers and get their feedback.

CloudViews had a long march towards getting the basic things working, including things like unit testing, integration testing, flighting, debugging, etc. Our first checkpoint was to make CloudViews as optional feature in an upcoming SCOPE release, i.e., a private preview. Next, we focussed on making it real for a pilot customer, discovering and fixing several bugs along the way. Once we gained more confidence, we announced it as public preview it in the next release. This was followed by further stabilization and getting it working for more customers, eventually leading to general availability (GA). Finally, there was a march towards getting customers on-boarded and addressing their adoption pain. All along we learned valuable lessons.

6. QUANTIFYING IMPACT

Now that the first version of the feature is built, it is time to market your work. The easiest way to sell something is by providing quantitative metrics, e.g., by validating the feature on a subset of customer data. Most companies have a way to do A/B testing before releasing new products. Get a hold of those testing tools and estimate the wins on a subset of real customer data. It may very well happen that the wins are not exactly the same as the projected wins from the initial workload analysis. Don't be defensive. Instead, analyze the gap between the expected and the actuals, and reconcile with it. Also, try to think ahead on what the observations translate into: are there more roadblocks that one should anticipate? is this leading to other interesting problems for the future? are there some fundamental properties about the problem space that could be distilled out? These may help plan some of the next steps going forward.

Once there are observed wins on one subset of the customer data, use that to motivate and drive adoption by all other customers. Note that customer adoption of any new product could often be a long journey and PMs are often the best buddies in this journey; work with them closely. In fact, after the first few instances, the PMs should be completely owning (as well as credited henceforth) for driving the customer adoption. In this phase, there will also be the inevitable cycle of bugs-fixes, releases, and usage. So, this is the time to harden the feature for production readiness. And given that multiple people will be helping with these different stages, be very open to share ownership, credits, and accolades. In particular, work actively towards making everyone involved visible and rewarded.

We took several steps to capture the impact of CloudViews. First of all, we added the tooling to make the feature visible to the end users. Then, we added telemetry to log the use of the feature for offline analysis. Finally, we ran reporting scripts to analyze and compare the query logs before and after the feature was enabled, and cross channeled them to drive further adoption.

7. PUBLISHING

There is a tremendous value in publishing applied research work. First of all, it is important to develop a sense of clarity around the key ideas. The process of writing forces one to convey things in a crisp, concise, and understandable manner, which alone is very useful for communicating with peers, collaborators, or anyone interested in the work. Ideas well communicated are

long lasting and have greater impact. Well written ideas are also well thought through and serve as a good basis when preparing for a presentation. In fact, presentation quality improves dramatically if it is based on a published work. Second, having a work reviewed and accepted for publication in a journal/conference gains credibility. Therefore, it is also important to target top journals/conferences in order to get higher credibility. Published works are easier to discover and reference to, making the researchers more credible over time. Third, it is desirable to have visibility, both for individuals and the team. Personal visibility can help people stay motivated, grow their career, and build future collaborations. Team visibility is needed for attracting top talent and for continued funding to the team, both essential for the team's survival. Fourth, it is useful to engage with the broader research community by sharing the results, the systems built, or just the directions pursued. These can be helpful to motivate industry relevant problems that other people can catch up or contribute. Wherever possible, consider sharing production data, or code, or customer insights to enable people benchmark and validate their solutions, or even derive newer problems.

Writing a paper from an industry research lab could turn out to be relatively easy. In fact, just writing down the previous six sections can help to motivate the problem, show data on why it is significant, discuss the ideas, describe the implementation, and finally show empirical evidence on how good it works. These steps backed by real problems, real data, and real workload can certainly go a long way in making the paper impactful. Consider both research and industry tracks — research tracks for problems where you have gone deeper and innovated on specific solutions, and industry track where you have considered the end-to-end scenarios, abstractions, or platforms, and their practicality in industry setting. Finally, be forthright in what is actually deployed in production and what is not. Apart from intellectual honesty, it is also important to not upset people from the product side with exaggerated claims, which the reviewers and editors have practically no way to verify before accepting the paper for publication.

Regarding authorship, consider that product teams may have built the system while not necessarily contributing to the writeup, and may therefore deserve authorship. As the paper draft develops, get feedback from product teams to be more precise about: (i) the prior state of things before the feature came around, (ii) how the feature was implemented, (iii) what is deployed and what is not. Involving product people generally improves the quality of the paper, makes it more detailed, and avoids feelings of discredit or misrepresentation.

In CloudViews, we found a new take on the older problem of materialized views, namely applying it to a cloud-based job service. As we began writing the CloudViews paper, we soon realized that there was too much material for a single paper, and so we split it into two papers, a system paper [8] and algorithm paper [6], and crediting the V-team in these papers was super helpful in boosting morale and keeping the motivation high.

8. SUCCESS METRICS

It is easy to see research output in binaries — whether it worked or whether it failed. Some people go even further to judge research output purely in terms of dollar amount. However, such a perspective misses the valuable lessons learned along the way, in each of the seven steps described before. For instance, a fresh understanding of product with the intent of working on it may reveal several gaps, analysis of product related data can discover exciting opportunities to make things better, getting ideas funded can surface the cost and feasibility of the effort (or even related efforts), teaming up with set of people can generate feedback on different aspects of the problem, going ahead and actually implementing the ideas could provide a good engineering experience, objectively measuring the impact could validate whether the ideas would actual work or not, and trying to get things published can tell how much and why the external world cares about these ideas. All of these intermediate lessons are equally important and should be incorporated when defining the success metrics. In fact, success metrics should also equally embrace the negative results, since not everything is going to be a success, particularly in a research team. And so people should be incentivized to still take measured amount of risks and report back the negative results, if any. Once the success metric is agreed upon, it is important to use that to evaluate the project. A good time to do that is while trying to get the work published, since the project must be at an advanced stage by then. In fact, use the success metrics to prove the point in the publication.

The CloudViews project was fairly successful on all of the above steps, however, given the duration of the project over multiple years, it also helped us realize the value of each step along the way. This leads us to believe that the key question when evaluating the success of a research project is to check whether one or more of the above steps were completed or not, and those should be presented as success to the management for the individual rewards and career progression.

9. MOVING ON

It is important to wind down a project once the expected success metrics have been achieved. In fact, winding down a project is just as important as spinning it up, especially in a research lab where often the goal is to explore and expand on newer frontiers. But there are a

set of things that need to be done before winding down. First of all, product people need to be trained to be able to run, operate, and debug the feature independently. The researcher needs to remove herself from the critical path and transfer ownership to the product teams. This is an important step and the technology transfer is only truly complete when the show goes on even when the researcher is not around.

Moving on involves not just winding down the current project but also finding the next one. Consider expanding the scope of the current problem, or looking at related problems, or defining the next level of abstraction that becomes relevant beyond the current one. Think about the expertise built so far that could be leveraged or the unfair advantage that has been gained from the previous project. While shaping the next problem, based on the experiences from the previous ones, articulate it as the vision for the next few years. This is timely to reflect back on how things worked in the past, think ahead on how you would like them in the future, and motivate or influence the future directions of the team. There will also be an expectation to present the larger vision as one grows more senior in the team. Not to mention it will help validate the course of action and get feedback for any corrective measures, if needed.

Two followup directions emerged from the CloudViews project, namely, to apply computation reuse to other query engines, and to consider other workload optimizations for the SCOPE (and other) query engines. As a result, we started multiple efforts in both directions, e.g., applying CloudViews to the Spark engine [10] and learning cardinality models from SCOPE workloads [11]. We further abstracted our ideas into a common workload optimization platform, called Peregrine [7], to make it easier for developing workload optimization features for different cloud query engines.

10. PORTFOLIO MANAGEMENT

One of the last lessons that we found valuable for being effective at Microsoft is to gradually develop a portfolio of projects, in particular as one transitions from the first project to the next ones. Managing a portfolio has several key benefits. First of all, research projects generally have a longer end to end arc, from ideation all the way to production deployment and adoption, and different stakeholders are involved differently in each of the stages, e.g., there could be more role for the researchers in the early ideation stage, more role for software developers in the system building stage, and more role for program managers in production deployment and onboarding. When one project does not require active involvement in its current stage, having a portfolio helps to remain occupied in other ones.

A portfolio also allows to pipeline projects such that they reach subsequent stages in succession. This can help in sustaining productivity at workplace, which is important both for professional success and for personal happiness. Researching on a single idea for too long can also make things boring and dampen the creativity. Juggling more than one project or idea can keep things interesting and can even cross pollinate ideas.

A portfolio also helps explore alternate ideas that are hard to pursue in a one project regime. This not only keeps the curiosity burning but also provides room to pursue riskier bets, which may or may not see the light of the day, or hot new trends, which need timely attention to not lose the early bird advantage. In a way, considering projects with different magnitude of ambition and feasibility, e.g., crazy new idea that may be completely impractical versus a straight forward extension of the earlier work, also hedges the bets and improves the chances of success, both in terms of making incremental moves and shooting for groundbreaking shifts.

CloudViews lead us to a vibrant portfolio consisting of multi-query optimizations, ML for systems, and platforms for workload optimization. Such a portfolio sets us up for an interesting research landscape, where we hope to create both personal and team identities.

Acknowledgements. We would like to thank the CloudViews and SCOPE teams for their continuous feedback, support, and openness to take on riskier bets. We are grateful to Hiren Patel for championing the CloudViews project from the PM side. And finally, special thanks to the GSL management and Raghu Ramakrishnan for supporting the project all along.

11. REFERENCES

[1] CloudViews Project. https://www.microsoft.com/en-us/research/project/cloudviews/.
[2] Gray Systems Lab. https://azuredata.microsoft.com/labs/gsl.
[3] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou. Re-optimizing Data-parallel Computing. In *NSDI*, 2012.
[4] A. Agrawal, R. Chatterjee, C. Curino, A. Floratou, N. Gowdal, M. Interlandi, A. Jindal, K. Karanasos, S. Krishnan, B. Kroth, J. Leeka, K. Park, H. Patel, O. Poppe, F. Psallidas, R. Ramakrishnan, A. Roy, K. Saur, R. Sen, M. Weimer, T. Wright, and Y. Zhu. Cloudy with high chance of DBMS: A 10-year prediction for Enterprise-Grade ML. In *CIDR (to appear)*, 2020.
[5] N. Bruno, S. Jain, and J. Zhou. Continuous cloud-scale query optimization and processing. In *VLDB*, 2013.
[6] A. Jindal, K. Karanasos, S. Rao, and H. Patel. Thou Shall Not Recompute: Selecting Subexpressions to Materialize at Datacenter Scale. In *VLDB*, 2018.
[7] A. Jindal, H. Patel, A. Roy, S. Qiao, J. Yin, R. Sen, and S. Krishnan. Peregrine: Workload Optimization for Cloud Query Engines. In *SOCC (to appear)*, 2019.
[8] A. Jindal, S. Qiao, H. Patel, Z. Yin, J. Di, M. Bag, M. Friedman, Y. Lin, K. Karanasos, and S. Rao. Computation Reuse in Analytics Job Service at Microsoft. In *SIGMOD*, 2018.
[9] R. Levin. A Perspective on Computing Research Management. *Operating Systems Review*, 41(2):3–9, 2007.
[10] A. Roy, A. Jindal, H. Patel, A. Gosalia, S. Krishnan, and C. Curino. Sparkcruise: Handsfree computation reuse in spark. In *VLDB*, 2019.
[11] C. Wu, A. Jindal, S. Amizadeh, H. Patel, S. Qiao, W. Li, and S. Rao. Towards a Learning Optimizer for Shared Clouds. In *VLDB*, 2019.
[12] J. Zhou, N. Bruno, M.-C. Wu, P.-Å. Larson, R. Chaiken, and D. Shakib. SCOPE: parallel databases meet MapReduce. *VLDB J.*, 21(5):611–636, 2012.